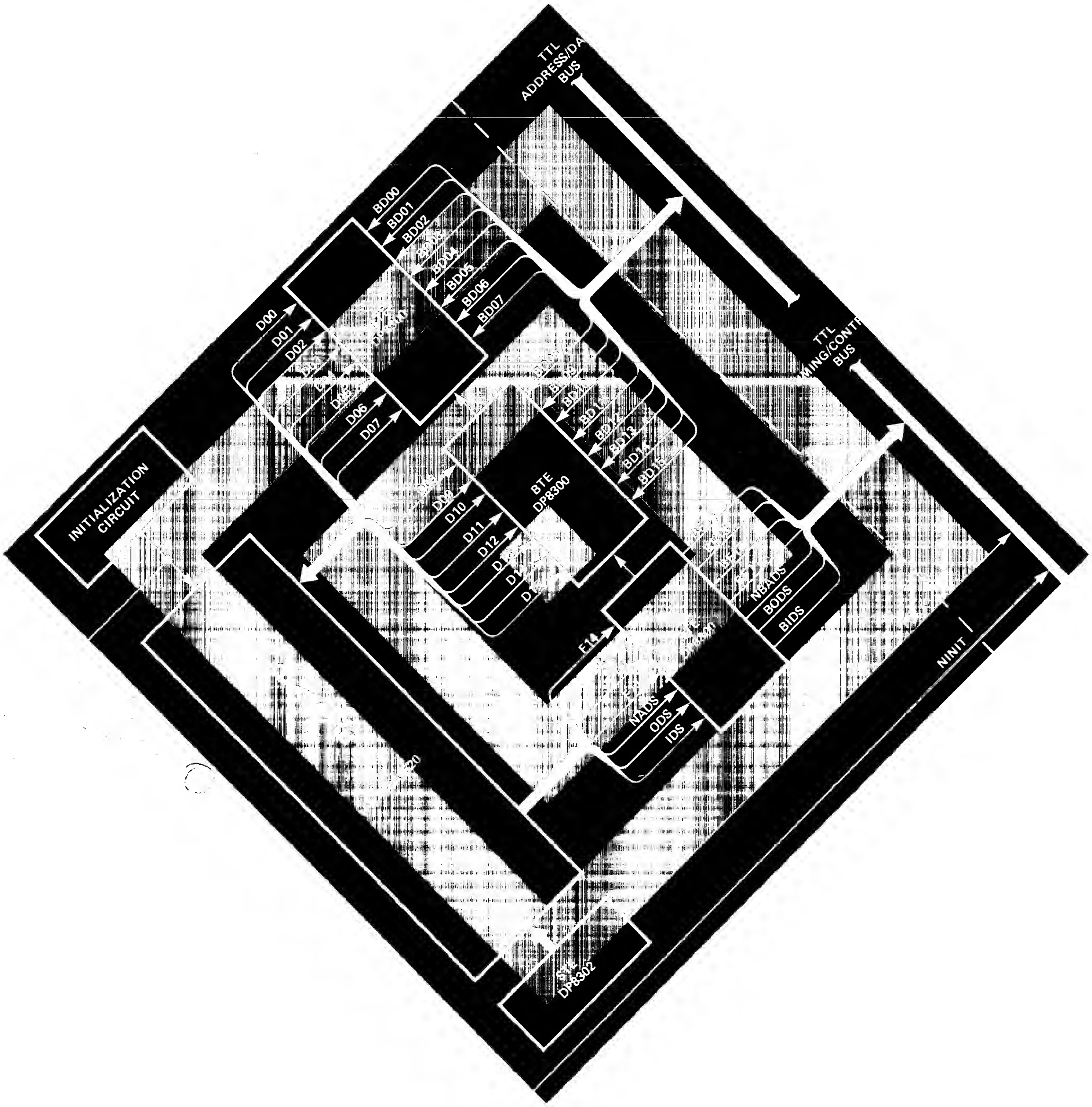
PACE Microprocessor System Design Manual

National Semiconductor



Publication Number 420305292-001A
Order Number IPC-16A/928
March 1977
$5.00

PACE Processing And Control Element

Microprocessor System Design Manual

PREFACE

The PACE System Design Manual serves as a source of basic design information. Included are descriptions of the PACE Microprocessor: its architecture, operational characteristics, instruction set, basic requirements for designing PACE-based systems, and both memory and peripheral interfacing techniques. The material has been presented in such a way as to provide design criteria not only for PACE but also to provide other pertinent design information that interrelates PACE to various other support and interfacing devices.

The material within this publication is up-to-date at the time of publication but is subject to change without notice.

Copies of this publication and other National Semiconductor publications may be obtained from the National Semiconductor sales office or distributor serving your locality.

Other PACE publications that may be of interest are listed below:

- PACE Logic Designers Guide to Programmed Equivalents to TTL Functions, Order Number IPC-16A/927. *Explains how standard TTL/MSI functions may be implemented by software using the PACE Microprocessor.*
- PACE Assembly Language Programming Manual, Order Number IPC-16S/969Y. *Provides tutorial and reference information required for writing user application programs. Includes detailed descriptions of the assembly language, source statements, programming techniques, and assembly input/output formats.*
- PACE Development System Users Manual, Order Number IPC-16P/108Y. *Comprises comprehensive descriptions of the PACE Development System, including both operational and functional information.*
- IPC-16A/520D MOS/LSI Single-Chip 16-Bit Microprocessor (PACE) Data Sheet. *Provides parametric specifications of PACE. (Also includes some of the functional data provided in this manual.)*
- PACE Microprocessor Disc Operating System (DOS) Users Manual, Order Number IPC-16P/840Y. *Comprises PACE DOS operational procedures for both equipment and software.*
- PACE Microprocessor Low Cost Development System Users Manual, Order Number IPC-16P/301Y. *Provides installation, operational, and interfacing information.*
- National Semiconductor Memory Data Book.

Various other National Semiconductor publications pertaining to PACE-based products are or will be available. Contact our local sales office or distributor for information.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

ILLUSTRATIONS

Chapter 1

AN INTRODUCTION TO PACE

1.0 INTRODUCTION

National Semiconductor's Process and Control Element (PACE) was the industry's first single-chip 16-bit microprocessor, and PACE continues to provide designers with the built-in sophistication that results in cost-effective systems and simplicity of application. As the microprocessor marketplace reaches maturity, some lessons learned during similar stages of minicomputer development are being rediscovered. Foremost among these lessons is an appreciation of the importance of word length: the inherent power and advantages of a 16-bit processing system over 8-bit systems are once again being realized. PACE is a true 16-bit microprocessor and, in addition to the innate superiority obtained from this fact, it incorporates a variety of other features that result in ease-of-use and uncluttered, economical implementation of powerful systems. The paragraphs that follow highlight the features of PACE and the resources that are available to support development of PACE-based systems. A detailed description of PACE is provided in chapter 2. Chapters 3, 4, and 5 provide guidelines and examples to assist in the design of PACE systems.

1.1 THE PACE MICROPROCESSOR

Some of the resources of PACE are illustrated in figure 1-1. PACE makes use of 16-bit data words and 16-bit instruction words, and features a powerful, efficient, and flexible set of 45 instruction types. The 16-bit instruction words greatly increase the power of PACE instructions beyond those of 8-bit processors. Further, the full 16-bit PACE-generated address words increase both efficiency and throughput over those of conventional 8-bit processors. This results from the elimination of multiple-precision operations usually required for 8-bit data elements, and the reduction of program memory overhead.

System memory requirements are further reduced by the four general-purpose working registers (accumulators AC0 – AC3) and a ten-word last-in-first-out (LIFO) stack provided by PACE. The accumulators reduce the number of memory load and store operations associated with saving temporary and intermediate results. The LIFO stack provides additional on-chip storage and further reduces program and data storage overhead while inherently reducing interrupt-response time (because it saves return addresses during interrupt servicing and subroutine execution). Thus, the accumulators and stack reduce memory overhead by providing on-chip storage, and increase system throughput by reducing the number of memory-access operations required.

Interfacing PACE to input/output devices is simplified by three data strobe signals that synchronize parallel data transfers, three sense inputs, and four flag outputs that can be used to implement device control functions and serial data transfers, and a multiple vectored priority interrupt structure for fast response to real-time control situations.

Parallel data transfers occur via the multiplexed address/data bus and, since PACE uses common memory input/output address space (memory-mapped input/output) the entire PACE repertoire of memory-reference instructions can be used to implement efficient and powerful input/output interfaces with a minimum of hardware. The Extend Signal further simplifies interfaces by permitting operation with slower devices with minimal penalties to system speed.

The three sense inputs (JC13, JC14, and JC15) are individually testable (along with 13 internal status sense conditions) by a conditional branch instruction in software and can be used for serial data input or to sense external system conditions. Serial data output can be accomplished using the four flag outputs (F11–F14): these software controlled outputs can be set, cleared, and pulsed by the user's program and are also useful for directly controlling system functions or setting software status indicators.

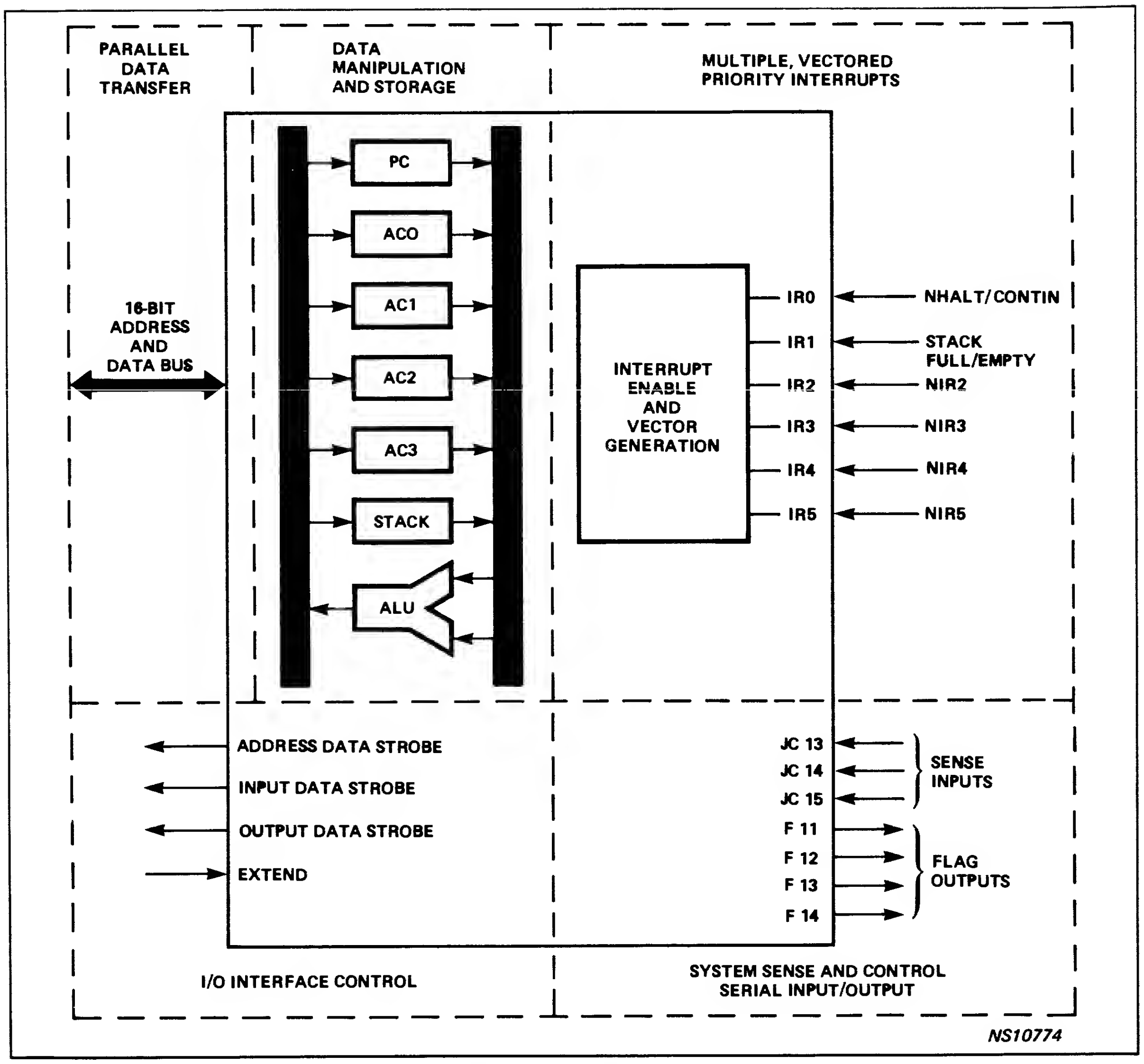


Figure 1-1. PACE Resources

The on-chip, 6-level, vectored, priority-interrupt structure provided by PACE, eliminates the need for any external, off-chip interrupt hardware. The interrupt vector feature provides automatic interrupt identification and thus eliminates program storage overhead and saves time – the time that other processors usually require for polling in order to identify the interrupting device. The interrupt system can be expanded easily by placing more than one peripheral on a priority level using a simple open-collector connection for a wired-OR input to an interrupt-request line.

This brief overview has highlighted some of the outstanding features of PACE. A fuller understanding and appreciation of the advantages and benefits derived from these PACE features can be gained from more-detailed discussions. Chapters 2 through 5 provide these detailed discussions along with useful examples and techniques to assist PACE system designers.

1.2 PACE DESIGN SUPPORT

The successful application of any microprocessor, including PACE, demands a variety of supporting products and services ranging from support chips, cards, and development systems to software, training, and field support.

PACE is fully supported throughout this spectrum of products and services. National Semiconductor's total support system ensures that the designer will have access to the tools he needs to efficiently implement his application. The paragraphs that follow describe the design support resources that are available.

1.3 PACE SUPPORT CHIPS

Three specially designed chips are available to simplify development of PACE systems. The System Timing Element (STE) DP8302 produces the MOS nonoverlapping clock signals required by PACE and also supplies two TTL clock signals to accommodate user requirements. The Bidirectional Transceiver Element (BTE) DP8300 is an 8-bit transceiver that provides controlled translation of signals between the PACE microprocessor MOS buses and the system TTL buses. The Microprocessor Interface Latch Element (MILE) DP8301 provides an 8-bit latched interface between the System TTL Address/ Data Bus and the user peripherals.

Figure 1-2 illustrates the relationship of the STE and BTE to PACE. The STE requires application of +5 and -12 volts and the addition of only an external crystal to produce the MOS and TTL clock signals. Three BTEs provide a fully buffered PACE CPU. One BTE connected in a driver-only mode to PACE permits

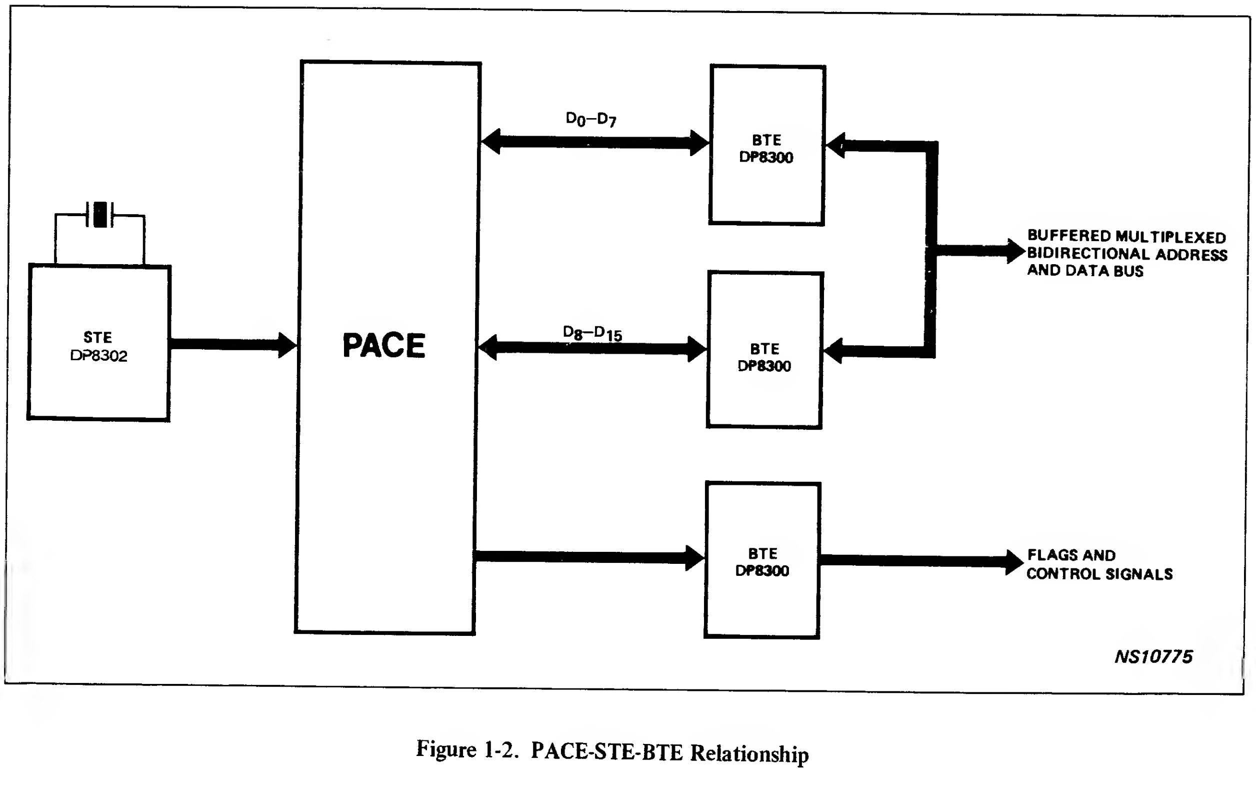

STE
DP8302
PACE
D0–D7
D8–D15
BTE
DP8300
BTE
DP8300
BTE
DP8300
BUFFERED MULTIPLEXED BIDIRECTIONAL ADDRESS AND DATA BUS
FLAGS AND CONTROL SIGNALS
NS10775

Figure 1-2. PACE-STE-BTE Relationship

buffering for the System TTL Timing and Control Bus, thereby providing seven TTL control signals and flags for distribution throughout the system. Two BTEs operate bidirectionally to buffer the 16-bit, time-multiplexed Address/Data Bus. Use of the STE and BTEs with PACE is described in detail in chapter 3. Chapter 5 includes a detailed discussion of system implementation of the MILE.

Because of the full complement of control signals provided by PACE, no other special components are required in the design of PACE systems. A variety of widely available, standard memory components can be used. Chapter 4 lists some of the more-popular memory devices and provides examples to illustrate the simplicity with which PACE can be interfaced to memory.

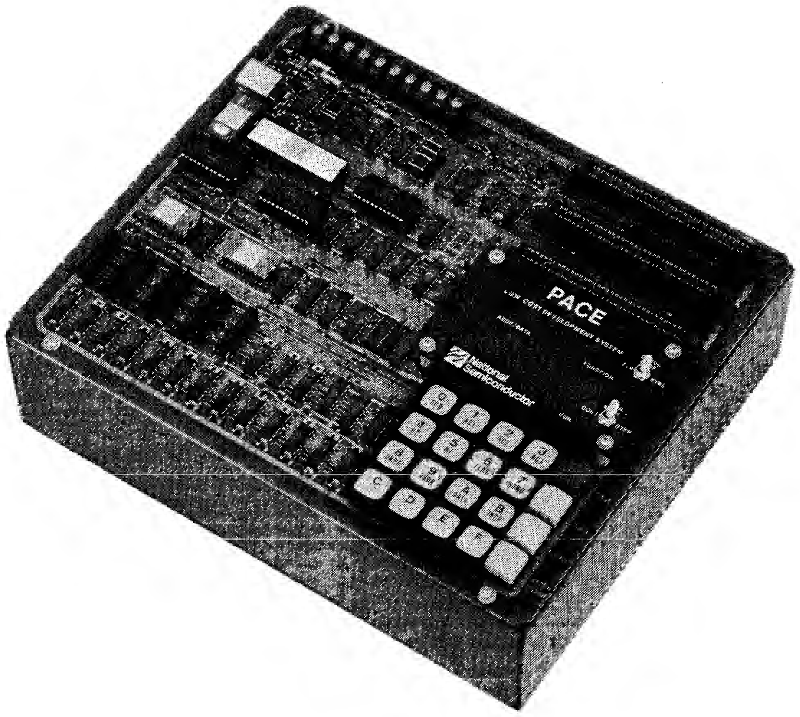
1.4 DEVELOPMENT SYSTEMS

Three PACE systems are available to simplify and speed the design and development of PACE-based application systems.

The PACE Low-Cost Development System (LCDS), shown in figure 1-3, allows the user to check out hardware and interface designs in a real-time environment, and can also be used to build software packages and convert them to firmware. The LCDS requires only the addition of a power supply to be fully operational, and the system can be expanded by connection of a TTY or RS-232-compatible terminal to the interface circuits provided on the LCDS. On-card ROMs contain a debug program and the input/output subroutines for communication with an optional terminal. Jumper options allow baud rates of 110, 150, 300, or 1200 bits-per-second to be selected. Further system expansion is easily accommodated using the three bussed sockets that accept compatible PACE Memory Application Cards and user-designed interface cards. A Cable Card is also available to extend the system bus to an external card cage.

The IPC-16P PACE Development System, shown in figure 1-4, is intended primarily to simplify the development of applications software. This system is a full-featured microcomputer and includes a comprehensive front panel to monitor and control system operation. A full array of peripheral options are available to extend the power of this system, and memory can be expanded from the standard configuration of 8K 16-bit words up to 32K 16-bit words. A powerful resident software package (described in 1.5) is provided to expedite the development of the user's application programs.

The PACE Disc Operating System (DOS) provides all the capabilities of the IPC-16P and further increases throughput and flexibility by providing the convenient mass storage capability of dual floppy disc drives. Included with the system is a DISC/CRT Interface Card that provides interfacing between the microprocessor, the dual disc drives, and a user-provided, optional CRT terminal. A DOS Monitor System is provided on a Read-Only-Memory (ROM) card and controls system configuration and linkage to other software subsystems. Also included on the ROM card is a File Input/Output Subsystem that provides the peripheral communication required by the remainder of the PACE DOS. A powerful disc-resident software package (described in 1.5) is included on a single diskette and completes the total-system capability obtained through use of the PACE DOS.



NS10737

Figure 1-3. PACE Low Cost Development System (LCDS)

NS10414

Figure 1-4. PACE Microprocessor Development System (IPC-16P)

1.5 PACE SUPPORT SOFTWARE

PACE is supported by a full complement of software provided to meet the needs of designers at every level of product development.

A BASIC interpreter has been developed for use with PACE and allows programs for design feasibility studies to be generated quickly and easily. This enhanced version of BASIC features real-time input/output and calls to assembly language subroutines. The BASIC interpreter is currently available for use with the PACE Development Systems and soon will be available for use with the PACE LCDS.

A Macro Assembler is also available and this powerful software tool permits extremely efficient assembly language programs to be generated with maximum ease. The Macro Assembler is memory resident in the IPC-16P and is provided on diskette with the PACE DOS Microprocessor Development System.

PACE DOS also includes a disc-resident File Manager Program that relieves the user of burdensome housekeeping tasks. The File Manager handles disc sector assignment automatically, and provides easy control of file maintenance and execution, space allocation, and file protection.

For users who do not have access to a PACE Development System, two cross assemblers are available. One version of the cross assembler runs on an IMP-16P or an IMP-16L Development System. A FORTRAN Cross Assembler is also available and allows programs to be developed on a variety of different host computers, including 16-bit minicomputers.

In addition to these primary software tools, a comprehensive array of editors, loaders, debug programs, and diagnostics are available to meet the needs of PACE system designers.

1.6 SUPPORT SERVICES AND PERSONNEL

A design effort does not end with the arrival of PACE hardware and software – it is just beginning. Recognizing this, National Semiconductor's support of PACE begins with training and predesign consultation and continues throughout your design cycle and beyond.

Microprocessor Training Centers, located in Miami, Florida and Santa Clara, California provide an intensive exploration of microprocessor applications with a good mix of hardware/software theory and hands-on laboratory experience. Beyond the classroom, applications engineers are available throughout the world and at our home base to help you analyze your specific application and translate your needs into a viable hardware/software configuration.

Consultation is available from the design concept stage through delivery of your final product and continuing education is available from COMPUTE (Club of Microprocessor Programmers, Users, and Technical Experts), an informal organization sponsored by National Semiconductor, dedicated to the distribution of ideas and techniques relating to the use of microprocessors.

Chapter 2

PACE MICROPROCESSOR DESCRIPTION

2.1 GENERAL DESCRIPTION

The PACE (Processing and Control Element) is a single-chip, 16-bit microprocessor packaged in a standard 40-pin package. PACE is a true 16-bit central processing unit; it makes use of 16-bit instruction words and 16-bit data words, and features a powerful, efficient, and flexible set of 45 instructions. All instructions use a single-word, 16-bit format – thus reducing memory accesses and program storage requirements.

2.1.1 PACE Architecture

The architecture of PACE (shown in figure 2-1) features a number of resources to minimize system program and read/write storage, to increase throughput, and to reduce the amount of external support hardware. PACE provides all the control and timing signals required for system or subsystem operation; these signals are described in detail in the sections that follow. Powerful and flexible data manipulation capabilities, the primary function of the central processing unit, and extensive internal data storage capacity that reduce the number of input/output cycles are also provided by PACE.

Data transfers between PACE (see figure 2-1) and memory or peripheral devices are effected over the 16-bit (D00-D15) parallel Input/Output Data Bus. The Input/Output Data Bus interfaces with the Instruction Register and the Operand Bus by way of the I/O Data Buffers. The Operand Bus also interfaces with seven registers (Temporary Registers 1 and 2, Program Counter, and AC0 through AC3) and a 10-word Stack. The seven registers and the Stack are provided for data storage. Four of the registers (AC0 through AC3) are available to the programmer as general-purpose accumulators. The Program Counter contains the address of the next instruction. The contents of any selected register or the Stack are routed over the Operand Bus to the Arithmetic and Logic Unit (ALU) and Shifter. Resultant ALU and Shifter output is returned to the selected register or Stack, as appropriate, by way of the Result Bus. The ALU and Shifter, besides performing arithmetic operations, also sets status flags in accordance with the data length (8-bit or 16-bit) selected by the state of the BYTE Status Flag.

All status information is stored in the 16-bit Status and Control Flags Register. The Status and Control Flags Register contents can be loaded onto the Operand Bus for temporary storage on the Stack or in any accumulator for examination or modification of status information.

Instructions under execution by PACE are stored in the Instruction Register and are interpreted and executed by a microprogram stored in an on-chip ROM. Instruction execution time is determined by the instruction under execution and memory access time. Appendix A provides an instruction summary, including the information required to calculate instruction execution times.

2.1.2 PACE Interface Signal Descriptions

PACE communicates with other system devices with its interface signals available via the 40 hardwired pins connected to PACE internal circuits. Three of the pins are used for input power and two are used for clock inputs. The remaining 35 pins are used to transfer 16-bit address and data information, to output control and timing synchronization information, and to accept control inputs that affect the operation of PACE.

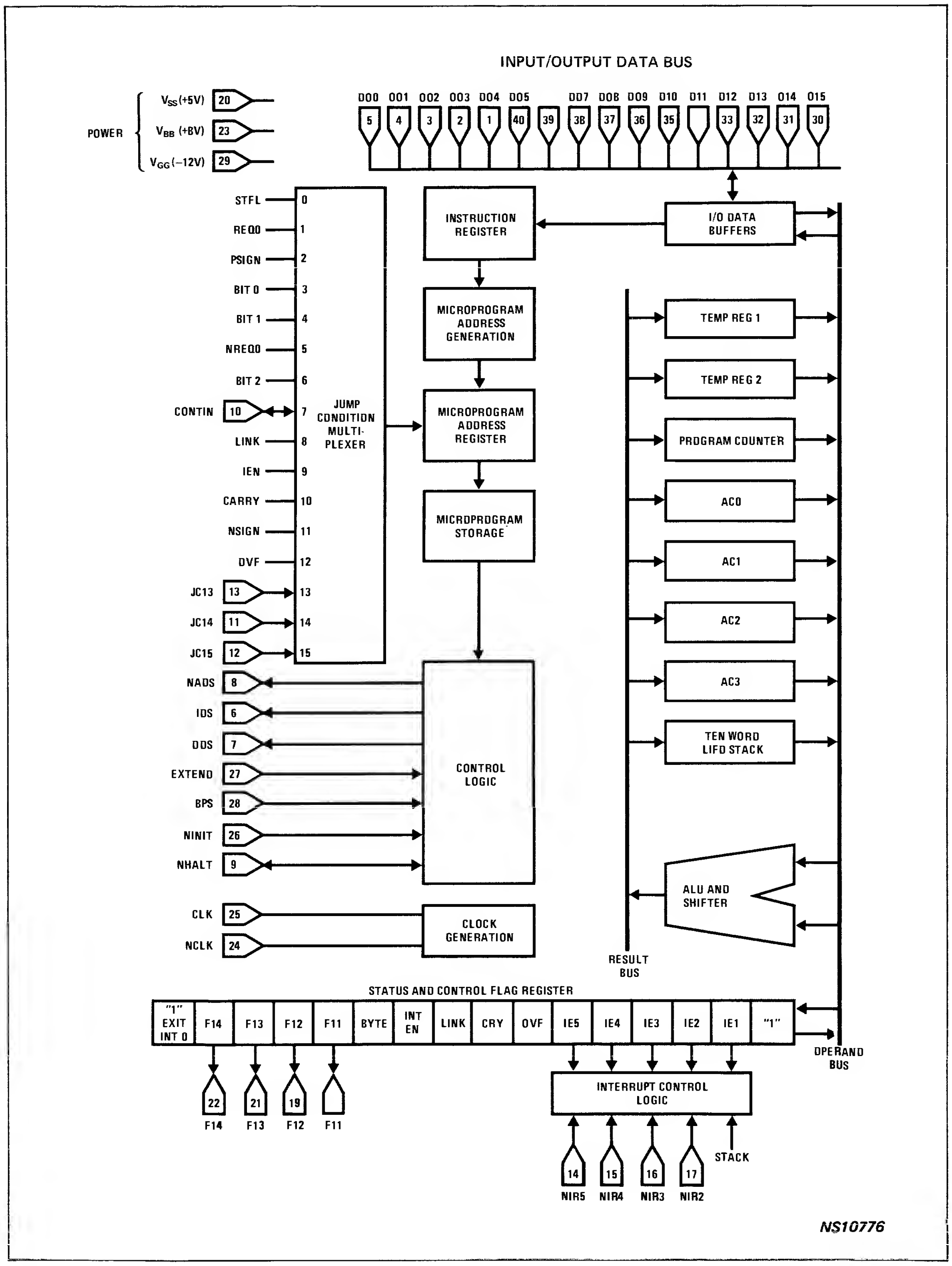


Figure 2-1. PACE Detailed Block Diagram

The pin assignments for PACE are shown in figure 2-2. Brief descriptions of the PACE interface signals are provided in table 2-1. Details on the use of the interface signals are provided later in this chapter and references to the appropriate sections are included in table 2-1.

NOTES

1. Positive logic convention is used throughout this manual. A logic '1' or high signal corresponds to a more-positive voltage level. A logic '0' or low signal corresponds to a more-negative voltage level. All signal names beginning with 'N' or followed by an asterisk (*) denote complemented signals that are asserted or activated by a logic '0'. Otherwise, signals are asserted by a logic '1'.
2. Bits are numbered from 00 to 15, right to left, with bit 00 representing the least significant bit.
3. The X' preceding a value denotes the hexadecimal numbering system.

2.2 PACE TIMING REQUIREMENTS

PACE requires single-phase, nonoverlapping true and complemented clock inputs as shown in figure 2-3. (Refer to PACE data sheet for detailed timing specifications.) The required timing inputs can be provided using the System Timing Element (STE) that is described in chapter 3 (3.1). PACE uses the external clock signals to generate internal multiphase clock signals that provide control timing for microprocessor operations. Four external time periods (t_p in figure 2-3) correspond to eight internal clock phases that comprise a single machine cycle of the PACE microprocessor. Figures 2-5 through 2-8 (which appear later in this chapter) illustrate the relationship between external clocks and internal clock phases. The execution of each PACE instruction requires four or more machine cycles, plus any required input/output cycle extends. (Appendix A provides an explanation of the method for computing actual execution times.)

2.3 INITIALIZATION

The PACE Initialize Signal (NINIT) input may be used at any time to initialize the microprocessor and should always be used after system power-up. Application of a low NINIT Signal clears the Stack Pointer, sets the flags to zero, sets the level-0 Interrupt Enable True, and sets the Program Counter contents to zero. The accumulators assume an arbitrary state. The NADS, IDS, and ODS data strobes are set false. Thus, if it is desired to initialize PACE during program execution, NINIT should be inhibited until after any data output cycle, that may be in progress, is completed. Inhibiting NINIT prevents erroneous data from being written to memory.

The minimum pulse width for NINIT is eight clock periods as shown in figure 2-4. The PACE data strobes (NADS, ODS, and IDS) are inactive for 16 clock cycles after the trailing edge of the NINIT Signal occurs. After the 16 clock cycles, the first NADS Signal occurs and the first instruction is accessed from memory location X'0000, unless a Level-0 Interrupt (Control Panel Interrupt) is present. All other interrupt levels are disabled.

Table 2-1. PACE Interface Signal Descriptions

Signal/Description	Reference Section	Signal/Description	Reference Section
CLK, NCLK (inputs) — External true and complemented clock inputs.	2.2	NIR2, NIR3, NIR4, NIR5 (input) — Interrupt Requests 2, 3, 4, and 5. When these negative-true input signals are low for 1 CLK period, minimum, the associated internal Interrupt Request Latch is set if the corresponding interrupt enable has been set by users program. The interrupt will be serviced after completion of current instruction if the Master Interrupt Enable is set. Interrupt Requests are prioritized, with NIR5 having lowest priority.	2.7 2.7.1
D00-D15 (input/output) — Data bus lines. Bidirectional MOS data lines used for input and output of data and for output of 16-bit addresses on I/O cycles.	2.4.1 2.4.2	NHALT (input/output) — Halt. When the negative-true NHALT signal is driven low by external logic, it effects microprocessor stall or 'Level-0' Interrupt, depending on timing of CONTIN signal. When not controlled by external logic, NHALT is driven low by PACE for 7/8 duty cycle while programmed halt condition exists. Programmed halt is initiated by the Halt Instruction and terminated by pulsing CONTIN line via external logic.	2.8
NADS (output) — Address Data Strobe. The negative-true NADS signal is sent out at the beginning of every data input/output cycle and indicates that a memory or peripheral address has been output on the data bus lines. The address is stable on the data bus while the NADS signal is active low.	2.4.1 2.4.2	CONTIN (input/output) — Continue. The CONTIN signal is used in input mode to terminate programmed halt, or to exercise microprocessor stall and Level-0 Interrupt, or as a jump-condition input that can be tested using a BOC instruction. In the output mode, CONTIN transmits an interrupt acknowledge pulse to acknowledge CPU response to an active interrupt input.	2.8
ODS (output) — Output Data Strobe. The ODS signal indicates to external circuits that the data bus contains valid output data.	2.42	BPS (input) — Base Page Select. The BPS signal enables one of two base-page addressing schemes to be selected. When BPS is low, first 256 words of memory constitute base page (page zero). When BPS is high, first 128 memory words and last 128 memory words constitute base page.	2.9.2
IDS (output) — Input Data Strobe. The IDS signal indicates to external devices that PACE is performing a data input cycle. The signal should be used by memory or peripheral devices to gate data onto the PACE data bus lines.	2.4.1	NINIT (input) — Initialize. While the negative-true NINIT signal is low, PACE operation is suspended, and IDS/ODS signals are set to inactive state. After NINIT completes low-to-high transition, the program counter is set to zero, the internal stack pointer is cleared, and all flags and interrupt enables are set low except Level-0 Interrupt Enable which is set high. All other registers contain arbitrary values.	2.3
EXTEND (input) — Extended Data Transfer. The EXTEND signal is used by slower memory or peripheral devices to temporarily increase time duration of data input/output transfers. The EXTEND signal should be driven high at the beginning of ODS or IDS signal and held high until output data has been captured or input data is made available to data bus. The EXTEND signal can also be used to suspend input/output operations by applying the signal after the end of ODS or IDS.	2.4.3	V_{SS} +5 Volts, V_{GG} −12 Volts V_{BB} V_{SS} +3 Volts (substrate bias)	
JC13, JC14, JC15 (input) — Jump Conditions 13, 14, and 15, JC13, 14, and 15 are user-specified inputs that can be tested using Branch-On-Condition (BOC) Instructions. If jump condition input specified in BOC Instruction is high, a program branch is effected. The JC13-JC15 signals are useful for testing status of external devices and receiving serial data.	2.6		
F11, F12, F13, F14 (output) — General-purpose control flag outputs from PACE Status and Control Flags Register. Individual flags may be set by Set Flag Instruction and pulsed or reset by Pulse Flag Instruction. The F11-F14 signals may be used for direct control of system functions or serial data output.	2.5		

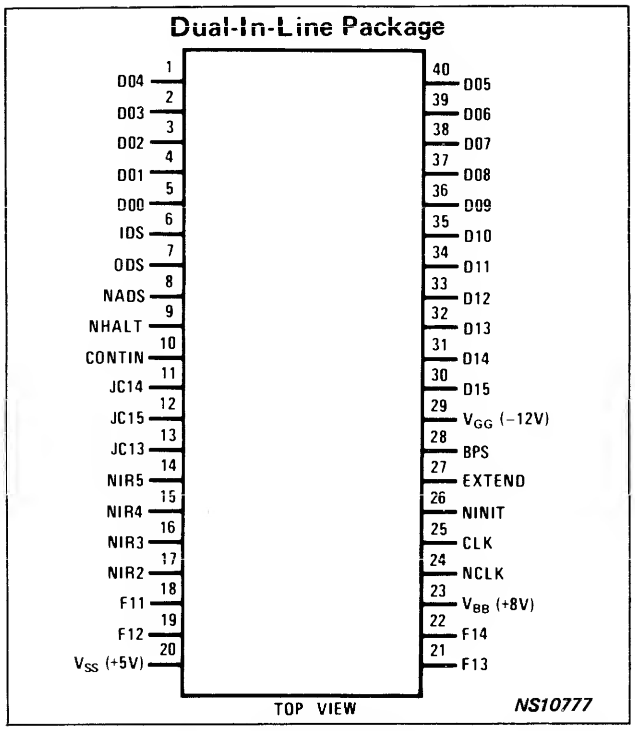
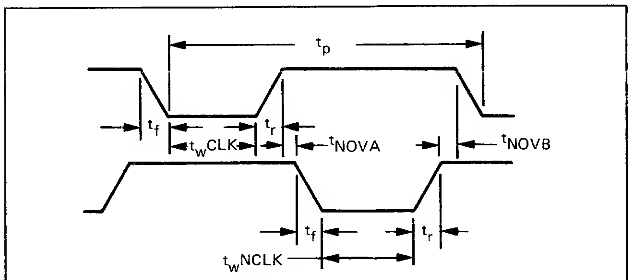


Figure 2-2. PACE Pin Assignments



where:
t_p = CLOCK PERIOD
t_{NOVA} = t_{NOVB} = CLOCK NONOVERLAP
t_{WCLK} = t_{WNCLK} = CLOCK WIDTH

NS10778

Figure 2-3. External Clock Timing

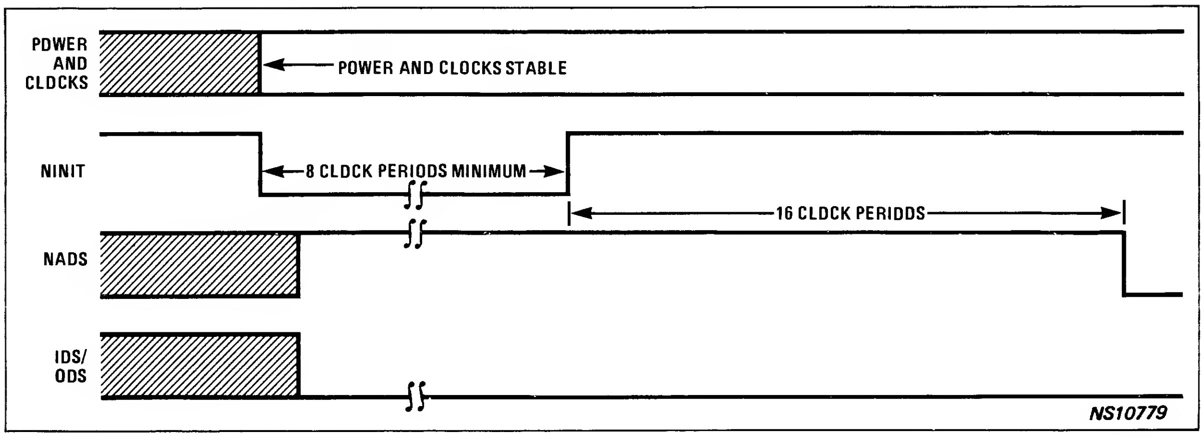


Figure 2-4. Initialization Timing

NOTE

If the NINIT input is held true (low) before power and/or clocks are totally stable, the NADS and NHALT outputs may have an undefined state for eight clock cycles after NINIT goes false (high). In order to initialize properly every time, NINIT should go true (low) *after* all the power supplies and clocks have stabilized.

2.4 DATA INPUT/OUTPUT OPERATIONS

The primary means PACE uses to communicate with external system devices is via 16-bit parallel data input/output (I/O) operations. The descriptions that follow apply both to memory devices (for either instruction fetch operations or data transfer operations) and to peripheral devices (using parallel data transfer techniques).

Parallel data transfers between PACE and external memory or peripheral devices take place over the 16 data lines (D00-D15). All input/output transactions consist of an address-output interval followed by a data-transfer interval. The data transfers are synchronized by the NADS, IDS, and ODS signals that are generated by PACE, and the EXTEND signal can be applied to PACE using external logic to increase the input/output cycle time.

2.4.1 Data Input Operation

Data input timing is shown in figure 2-5. Address data become valid one clock phase prior to the Address Data Strobe (NADS) and remain valid for one clock phase afterwards. Typically, NADS is used to strobe the address data into a latch, either internal or external to the system memory chips, or to clock decoded peripheral addresses into a flip-flop.

Following the output of address information, the output address drivers internal to PACE assume a high-impedance state and the Input Data Strobe (IDS) Signal goes high. The IDS Signal may be used to disable external output sense amplifiers and to enable input buffers. The IDS Signal remains high for three clock widths: the data lines must be driven to valid input data logic levels by the end of IDS.

Typically, the data-input timing allows operation of PACE in a system at maximum frequency if the system memory access time is less than two clock periods. If longer access time is required, the EXTEND signal may be used to increase the input/output cycle time. Use of the EXTEND signal is described in 2.4.3.

2.4.2 Data Output Operations

Timing for data-output operations is shown in figure 2-6. The address-output portion of the operation is identical to that described for data-input operations. Following the address, output data is placed on the data lines and, at approximately the same time, the Output Data Strobe (ODS) Signal goes high. The ODS Signal is used typically as a read-write signal for memory and as an output-data latch strobe for peripheral interfaces. Since output data remain valid following termination of the ODS Signal, the trailing edge of ODS can be used to clock the data into an external latch. The EXTEND signal (described in the paragraph that follows) can be used to extend the time that output data remain valid.

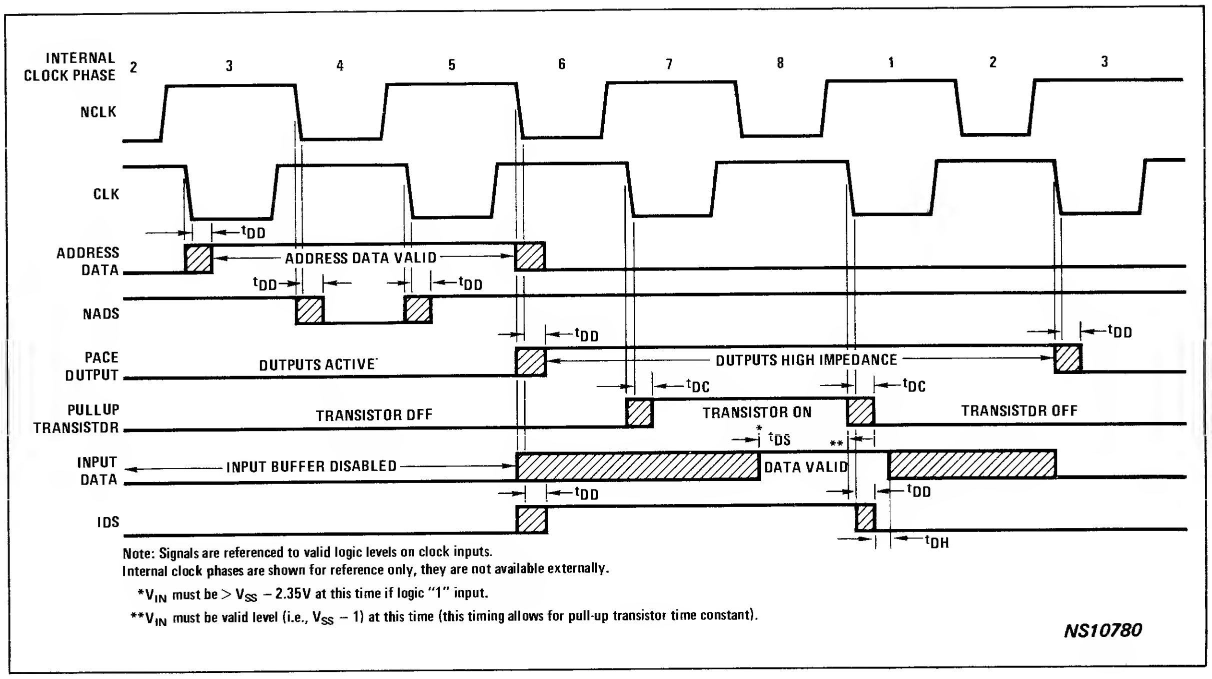

INTERNAL CLOCK PHASE
2
3
4
5
6
7
8
1
2
3
NCLK
CLK
ADDRESS DATA
ADDRESS DATA VALID
tDD
NADS
PACE OUTPUT
OUTPUTS ACTIVE
OUTPUTS HIGH IMPEDANCE
PULLUP TRANSISTOR
TRANSISTOR OFF
TRANSISTOR ON
TRANSISTOR OFF
tDC
INPUT DATA
INPUT BUFFER DISABLED
DATA VALID
tDS
IDS
tDH
Note: Signals are referenced to valid logic levels on clock inputs.
Internal clock phases are shown for reference only, they are not available externally.
*VIN must be > VSS − 2.35V at this time if logic "1" input.
**VIN must be valid level (i.e., VSS − 1) at this time (this timing allows for pull-up transistor time constant).
NS10780

Figure 2-5. Address Output and Data Input Timing

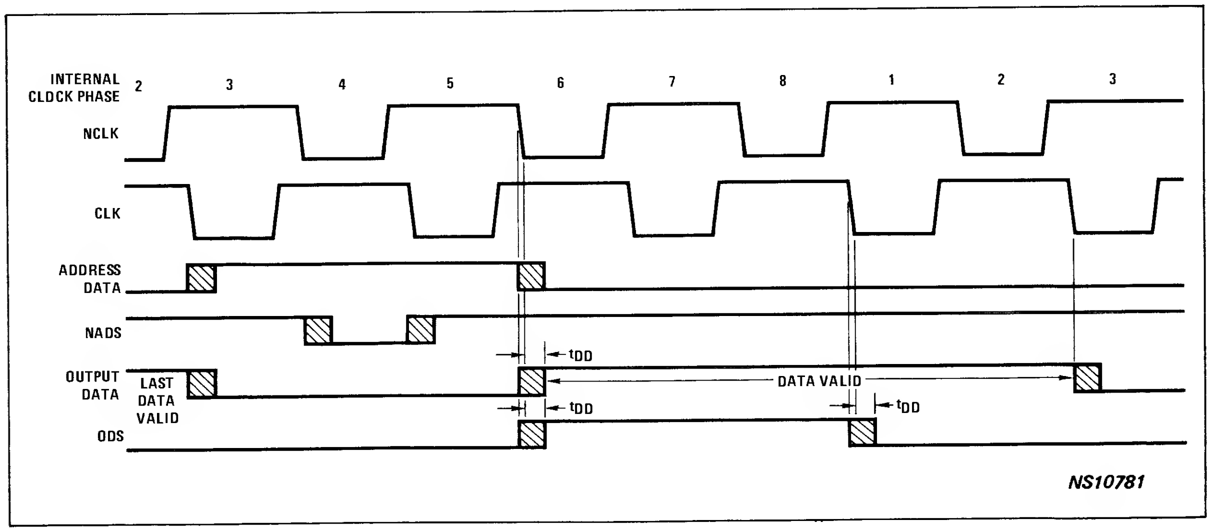


Figure 2-6. Data Output Timing

2.4.3 Use of EXTEND Signal

The EXTEND signal may be used either to increase the time duration of a data-input/output cycle or to suspend all PACE-initiated input/output operations.

Figure 2-7 shows the timing required when using EXTEND to lengthen the input/output cycle. The EXTEND signal may go high (true) during the address time or immediately after the start of IDS or ODS, but EXTEND must be true prior to the end of internal phase 6.

NOTE

If the EXTEND signal is not used, it should be tied to ground.

The timing shown in figure 2-7 provides the minimum extend of one clock period. Holding EXTEND true for n additional clock periods longer causes an extension of n + 1 clock periods. The EXTEND signal duration must not exceed the specified minimum refresh requirements of the PACE device. (Refer to appendix C for specifications.)

The EXTEND signal may also be used to suspend PACE input/output operations. This may be desirable in Direct Memory Access or multiprocessor systems to prevent input/output operations by PACE when the bus is in use by another device. Input/output operations are suspended by using the EXTEND signal immediately following an IDS or an ODS as shown in figure 2-8.

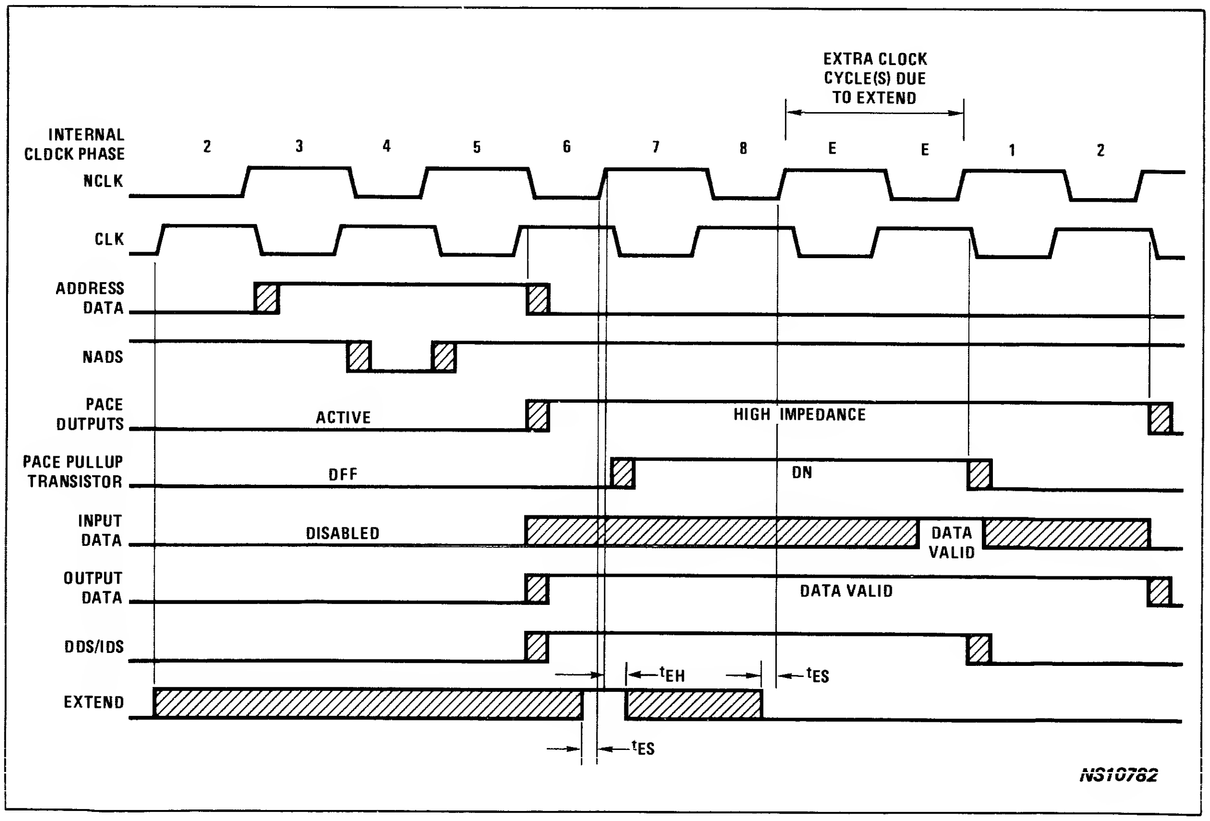


Figure 2-7. Extend I/O Signal Timing

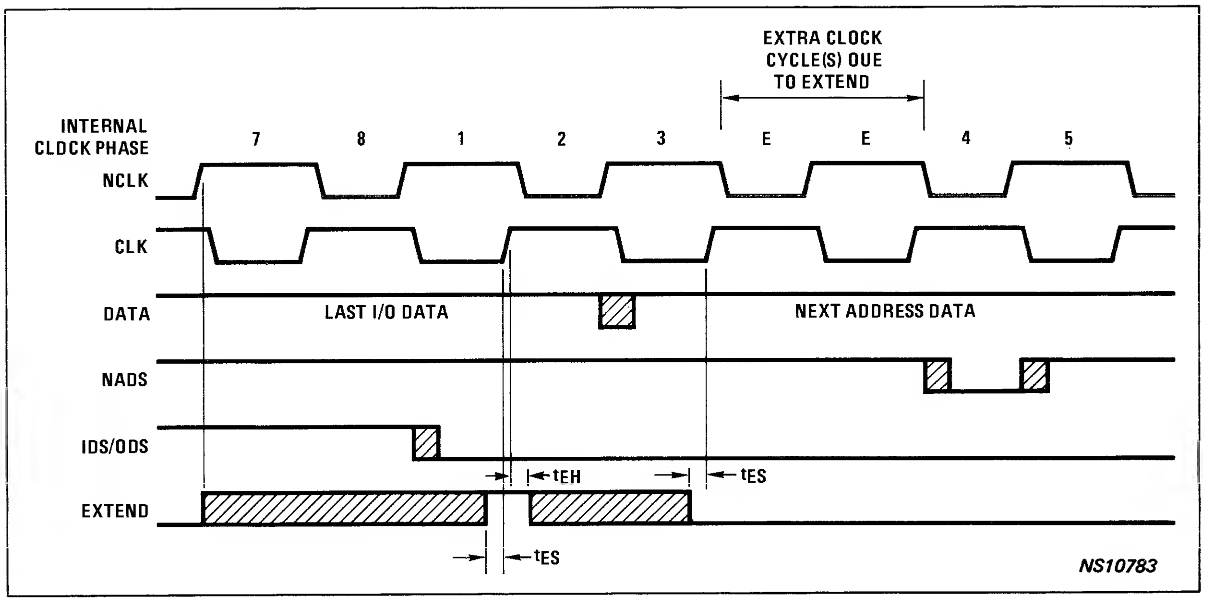


Figure 2-8. Suspend I/O Signal Timing

2.5 STATUS AND CONTROL FLAGS

Fourteen status and control flags are provided by the PACE microprocessor in the Status and Control Flag Register. The flags contained in this register can be accessed or restored as a 16-bit data word by using the Copy Flags to Register or Copy Register to Flags Instructions. Similarly, the contents of the register can be saved on the Stack or retrieved from the Stack. Individual flags can be set by using the Set Flag Instruction and reset or pulsed using the Pulse Flag Instruction. Four of the bits (F11, F12, F13, and F14) in the Status and Control Flag Register drive PACE output pins and may be used to directly control system functions or to accomplish serial data output. Table 2-2 provides descriptions of the individual status and control flags. Figure 2-9 shows the timing for Set Flag and Pulse Flag Instruction operations.

2.6 JUMP CONDITIONS

The PACE microprocessor contains a Jump Condition Multiplexer that samples the 16 jump conditions listed and described in table 2-3. The Branch-On-Condition Instruction (BOC) tests the output of the jump condition multiplexer. If the condition for branching (selected by the condition code specified in the BOC Instruction) is active, a branch occurs; otherwise, the next sequential instruction is executed. Note that 12 of the conditional jumps test conditions internal to PACE. The remaining four jump conditions (CONTIN, JC13, JC14, and JC15) are connected to PACE input pins and can be used to test external user-specified conditions. These jump condition inputs are also useful as serial data inputs.

NOTE

The CONTIN signal can also be used in conjunction with the NHALT signal to implement several special-purpose processor functions as described in 2.8.

Table 2-2. Descriptions of Status and Control Flags

Register Bit	Flag Name	Description	Flag Code (fc)
0	High ('1')	Bit 0 is not used and is always in logic '1' state. Referencing bit 0 with SFLG or PFLG Instruction has no effect. (May be used as NOP Instruction.)	0000
1 2 3 4 5	IE1 IE2 IE3 IE4 IE5	Flags IE1 through IE5 serve as Interrupt Enable Flags for five of six PACE interrupt levels. If Interrupt Enable is high and associated Interrupt Request occurs, microprocessor executes Interrupt Service Routine. If Interrupt Enable is low, associated Interrupt Request is ignored.	0001 0010 0011 0100 0101
6	OVF	Overflow Flag is set to state of twos-complement arithmetic overflow by arithmetic instructions. Overflow Flag is set high if sign bits (most significant bit) of two operands are identical and sign bit of result is different from sign bit of operands. If A, B, and R are sign bits of operands and result, then Overflow Flag is set according to equation $\mathrm{OVF} = (\overline{A} \cdot \overline{B} \cdot R) + (A \cdot B \cdot \overline{R})$ Sign bit is most significant bit for data length selected; thus, if data length is 8 bits, then bit 7 is sign bit; if data length is 16, then bit 15 is sign bit. State of OVF Flag is affected by instructions ADD, DECA, SUBB, RADD, and RADC.	0110
7	CRY	Carry Flag is set to state of binary or decimal carry output of adder by arithmetic instructions. Carry output is derived from most significant bit for data length specified by BYTE Flag. State of CRY Flag is affected by instructions ADD, DECA, SUBB, RADD, and RADC.	0111
8	LINK	Link Flag is included in shift and rotate operations as specified by Shift and Rotate Instructions. Link Flag is unaffected if not selected.	1000
9	IEN	Master Interrupt Enable Flag simultaneously inhibits all five of lowest-priority interrupt levels. No Interrupt Request is serviced unless individual Interrupt Enable Flag for associated Interrupt Request and master Interrupt Enable Flag are high. IEN Flag is set low every time any interrupt (except Level-0) is serviced. IEN Flag is set high by execution of Return To Interrupt Instruction (RTI).	1001
10	BYTE	BYTE Flag selects 8-bit data length when high and 16-bit data length when low.	1010
11 12 13 14	F11 F12 F13 F14	Flags 11 through 14 are general-purpose control flags. Flags 11 through 14 drive PACE output pins and may be used to directly control system functions.	1011 1100 1101 1110
15	High ('1')	Bit 15 is not functional and is always in logic '1' state. Addressing bit 15 with SFLG or PFLG Instruction sets the Level-0 Interrupt Enable high. The Level-0 Interrupt is described in Section 2.8.3.	1111

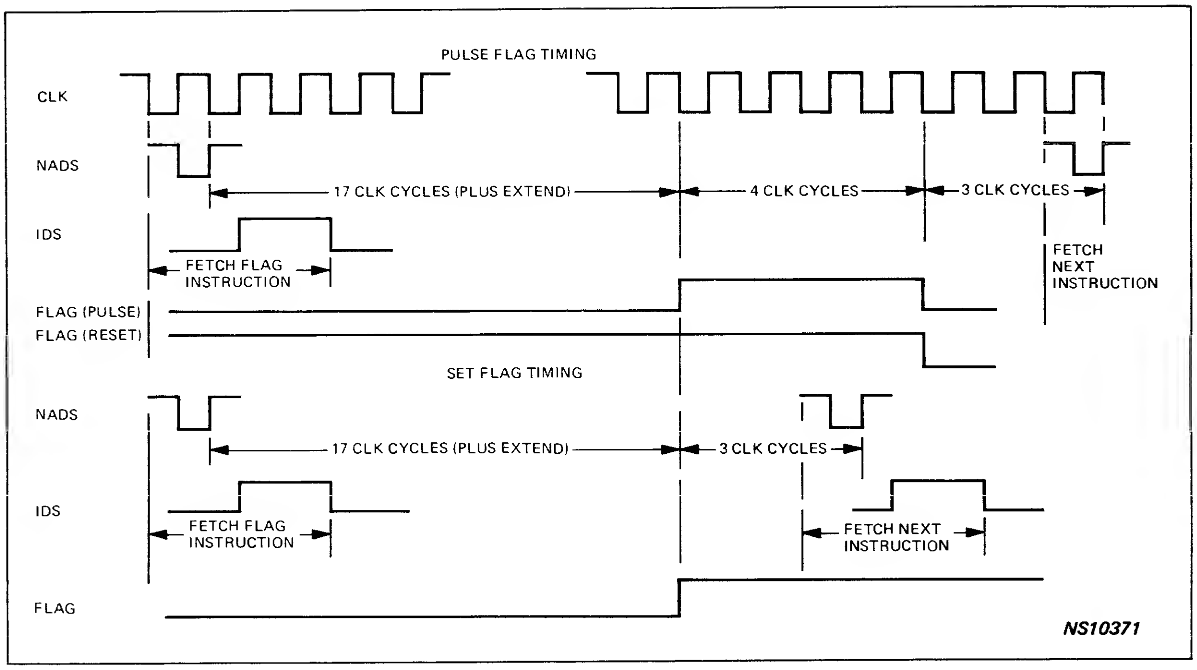


Figure 2-9. Pulse and Set Flag Timing Diagram

Table 2-3. Branch Conditions

Condition Code (cc)	Mnemonic	Condition
0000	STFL	Stack Full (contains nine or more words).
0001	REQ0	(AC0) equal to zero (see note 1).
0010	PSIGN	(AC0) has positive sign (see note 2).
0011	BIT0	Bit 0 of AC0 true.
0100	BIT 1	Bit 1 of AC0 true.
0101	NREQ0	(AC0) is nonzero (see note 1).
0110	BIT2	Bit 2 of AC0 is true.
0111	CONTIN	CONTIN (continue) Input is true.
1000	LINK	LINK is true.
1001	IEN	IEN is true.
1010	CARRY	CARRY is true.
1011	NSIGN	(AC0) has negative sign (see note 2).
1100	OVF	OVF is true.
1101	JC13	JC13 Input is true.
1110	JC14	JC14 Input is true.
1111	JC15	JC15 Input is true.

NOTES:
1. If selected data length is 8 bits, only bits 0 through 7 of AC0 are tested.
2. Bit 7 is sign bit (instead of bit 15) if selected data length is 8 bits.

2.7 INTERRUPT SYSTEM

PACE provides a 6-level, vectored, priority interrupt structure. This allows automatic identification of the priority level of an interrupting device and allows all devices on an interrupt level to be enabled or disabled as a group, independent of other interrupt levels. An individual interrupt enable is provided in the Status Register for each level, as shown in figure 2-10, and a master interrupt enable (IEN) is provided for all five lower priority levels as a group. The state of the Internal Interrupt Signal is tested by PACE during the Instruction Fetch Routine (internal to PACE) that is executed after completion of each instruction. Thus, if the Internal Interrupt Signal is high, the interrupt is automatically serviced.

NIR2 through NIR5 are referred to as user-specified interrupts; their operation is described in 2.7.1. The Level-1 Interrupt is dedicated for use as a stack-full/stack-empty interrupt and is described in 2.7.2. The Level-0 Interrupt is a special-purpose, nonmaskable interrupt and requires user manipulation of the NHALT and CONTIN signals; implementation of the Level-0 Interrupt is described in 2.8.3.

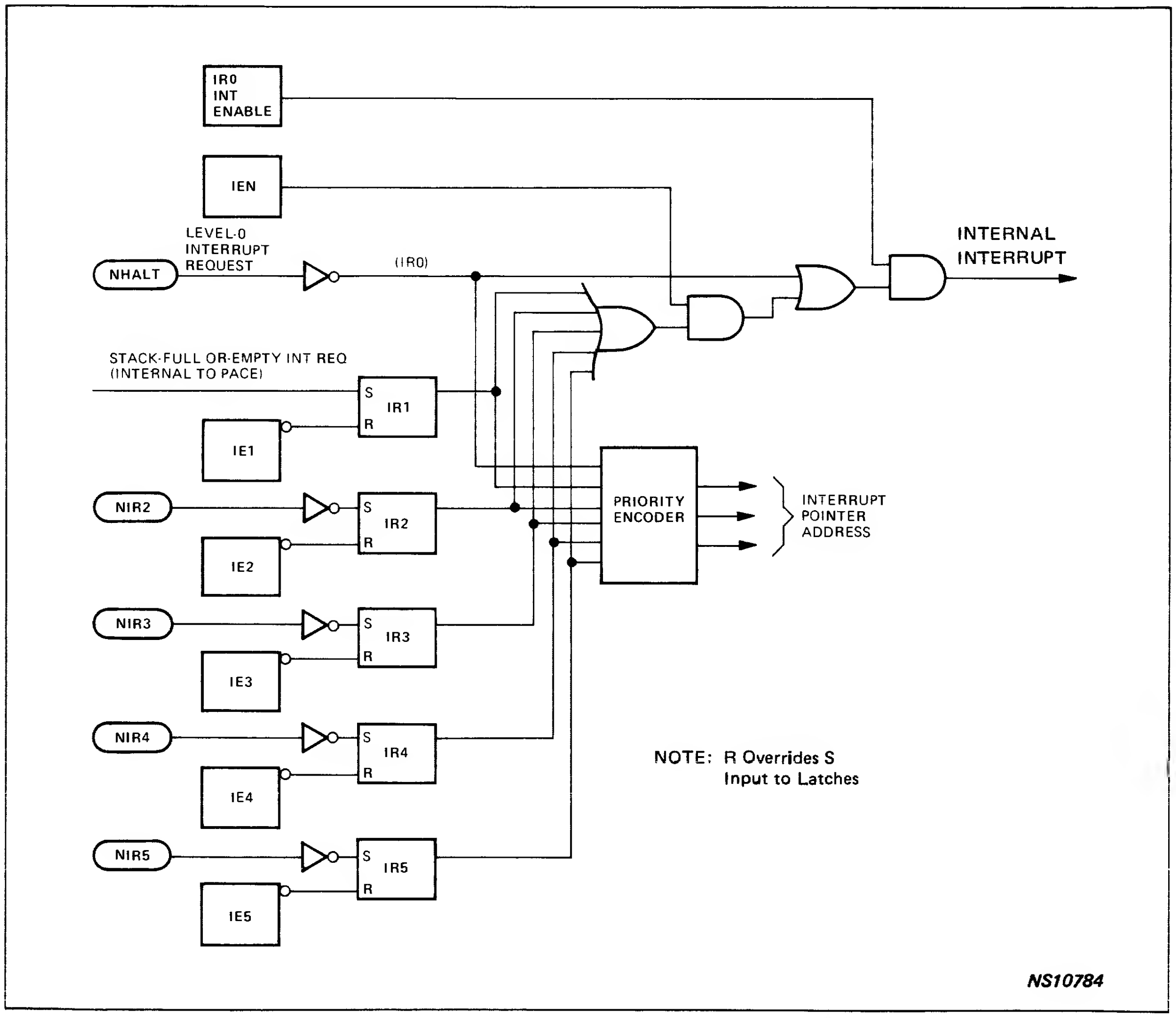

IR0 INT ENABLE
IEN
LEVEL-0 INTERRUPT REQUEST
NHALT
(IR0)
INTERNAL INTERRUPT
STACK-FULL OR-EMPTY INT REQ (INTERNAL TO PACE)
S
R
IR1
IE1
NIR2
IR2
IE2
NIR3
IR3
IE3
NIR4
IR4
IE4
NIR5
IR5
IE5
PRIORITY ENCODER
INTERRUPT POINTER ADDRESS
NOTE: R Overrides S Input to Latches
NS10784

Figure 2-10. PACE Interrupt System

2.7.1 User-Specified Interrupts

Negative-true Interrupt Request Inputs (NIR2 through NIR5) are provided to allow several interrupts to be wire-ORed to each input. When an Interrupt Request occurs, the associated Interrupt Request Latch (IR1 through IR5) is set if the corresponding Interrupt Enable Input is true. Since the Interrupt Request Latch can be set by any pulse exceeding one clock period, narrow timing or control pulses can be captured. If IEN is high, then an interrupt is generated and acknowledged after completing the current instruction. The interrupt acknowledge is provided by PACE on the CONTIN pin; refer to figures 2-11 and 2-12 and associated text for an explanation of the interrupt-acknowledge function of the CONTIN signal.

During the interrupt sequence, an address is provided by the output from the priority encoder. The address is used to access the Interrupt Pointer for the highest-priority Interrupt Request (IR0 has highest priority; IR5 has lowest priority). The Interrupt Pointers are stored in memory locations 2 through 8 (see table 2-4) for Interrupt Requests 1 through 5 and 0, respectively. The Interrupt Pointer specifies the starting address of the user-supplied Interrupt Service Routine for the particular interrupt level, except in the case of the Level-0 Interrupt (IR0), which is used primarily for alarm interrupts and Control Panel implementation (refer to 2.8.3 for details).

Before execution of the Interrupt Service Routine, the contents of the Program Counter are pushed onto the Stack and IEN is set low (false). This interrupt handling requires 28 clock cycles. The Interrupt Service Routine may set IEN high (true) after turning off the Interrupt Enable for the interrupt level currently being serviced (or resetting the Interrupt Request). The Interrupt Enable Flags can be set by the Set Flag (SFLG) and reset by the Pulse Flag (PFLG) Instructions. The Copy Register to Flags (CRF) Instruction can also be used to set or reset Interrupt Enable Flags. If an Interrupt Enable Flag is set or reset, one more

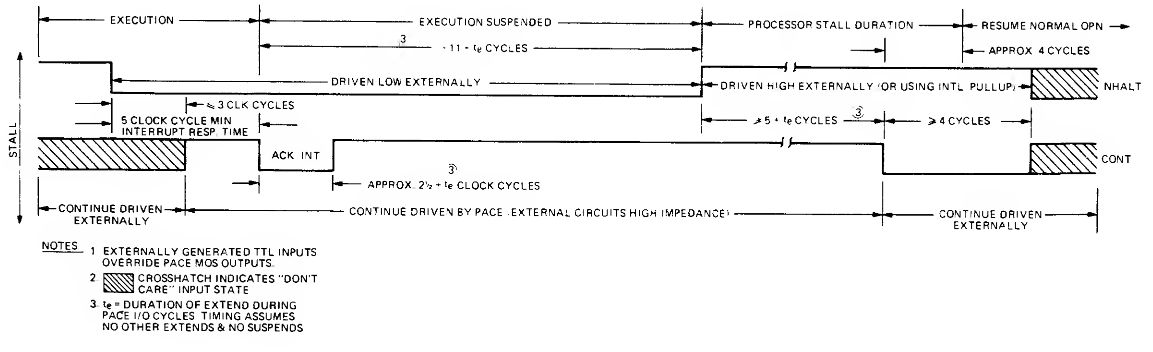

EXECUTION
EXECUTION SUSPENDED
PROCESSOR STALL DURATION
RESUME NORMAL OPN
3
11 + t_e CYCLES
APPROX 4 CYCLES
DRIVEN LOW EXTERNALLY
DRIVEN HIGH EXTERNALLY (OR USING INTL PULLUP)
NHALT
≤ 3 CLK CYCLES
5 CLOCK CYCLE MIN INTERRUPT RESP. TIME
≥ 5 + t_e CYCLES
3
≥ 4 CYCLES
STALL
ACK INT
CONT
3
APPROX. 2½ + t_e CLOCK CYCLES
CONTINUE DRIVEN EXTERNALLY
CONTINUE DRIVEN BY PACE (EXTERNAL CIRCUITS HIGH IMPEDANCE)
CONTINUE DRIVEN EXTERNALLY
NOTES
1 EXTERNALLY GENERATED TTL INPUTS OVERRIDE PACE MOS OUTPUTS.
2 ▨ CROSSHATCH INDICATES "DON'T CARE" INPUT STATE
3 t_e = DURATION OF EXTEND DURING PACE I/O CYCLES. TIMING ASSUMES NO OTHER EXTENDS & NO SUSPENDS
NS10480

Figure 2-11. Timing Diagram for Processor Stall using NHALT and CONTIN Signals

instruction is executed before the interrupt is enabled or disabled. The Return From Interrupt Instruction (RTI) also may be used to set IEN true. In this case, there is no delay and a pending interrupt takes place immediately after execution of RTI.

NOTE

The use of PFLG or CRF Instructions to disable the IEN flag allows one more instruction to be executed before the interrupts are disabled. If an interrupt should occur during execution of the PFLG or CRF Instruction, the subsequent use of RTI would leave IEN true (one) after the execution of PFLG IEN. To prevent this situation, the BOC Instruction may be used to test PFLG or CRF Instruction as follows:

```
PFLG IEN          ; TURN OFF IEN
BOC IEN, .-1      ; IS IEN FALSE?
                  ; YES
```

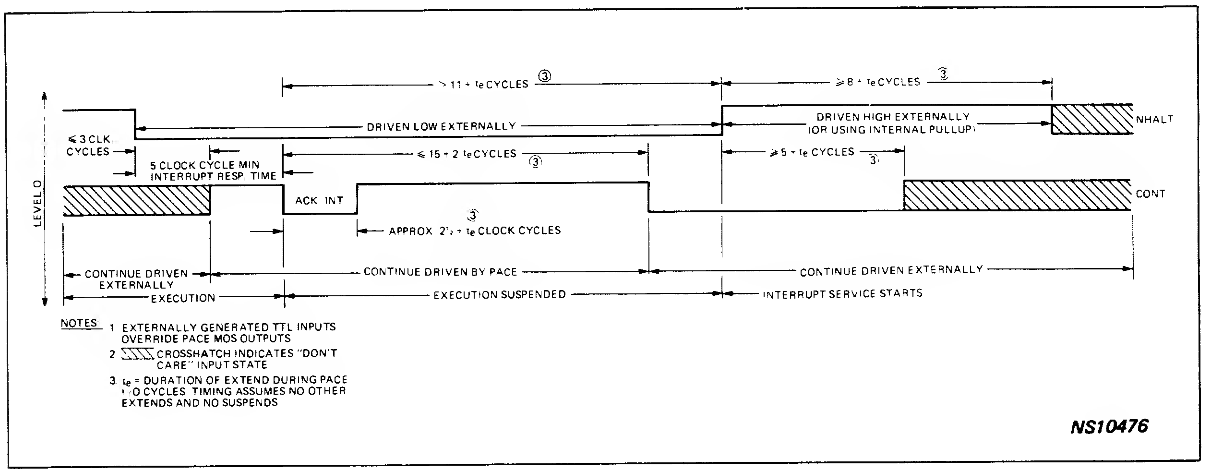

> 11 + te CYCLES ③
≥ 8 + te CYCLES ③
DRIVEN LOW EXTERNALLY
DRIVEN HIGH EXTERNALLY (OR USING INTERNAL PULLUP)
NHALT
≤ 3 CLK. CYCLES
5 CLOCK CYCLE MIN INTERRUPT RESP. TIME
≤ 15 + 2 te CYCLES ③
≥ 5 + te CYCLES ③
LEVEL 0
ACK INT
CONT
APPROX 2½ + te CLOCK CYCLES ③
CONTINUE DRIVEN EXTERNALLY
CONTINUE DRIVEN BY PACE
CONTINUE DRIVEN EXTERNALLY
EXECUTION
EXECUTION SUSPENDED
INTERRUPT SERVICE STARTS
NOTES
1 EXTERNALLY GENERATED TTL INPUTS OVERRIDE PACE MOS OUTPUTS
2 ▨ CROSSHATCH INDICATES "DON'T CARE" INPUT STATE
3. te = DURATION OF EXTEND DURING PACE I/O CYCLES. TIMING ASSUMES NO OTHER EXTENDS AND NO SUSPENDS
NS10476

Figure 2-12. Relative Timing for Level-0 Interrupt Generation

Table 2-4. Locations of Interrupt Pointers

Interrupt Pointer	Memory Location
Interrupt-0 Program	8
Interrupt-0 PC	7
Interrupt 5	6
Interrupt 4	5
Interrupt 3	4
Interrupt 2	3
interrupt 1	2
Not Assigned	1
Initialization Instruction	0

It should be recognized that the function of the individual Interrupt Enables IE1-IE5 is to arm or disarm the Interrupt Request Latch, whereas the function of the Master Interrupt Enable (IEN) and Interrupt Enable IR0 is to enable or disable the latched Interrupt Request lines.

2.7.2 Stack Interrupts

The response to a Stack Interrupt (Interrupt Level-1) is the same as described for user-specified interrupts. The initiation of the interrupt request, however, is internal and occurs automatically when a stack-empty or stack-full condition exists. The Stack Interrupt consists of a pulse applied to the set input of Interrupt Request Latch 1 (see figure 2-10). The pulse sets the latch if the IEN1 Flag is true; otherwise, the pulse is ignored. The Stack is implemented with a RAM and a Pointer, which can access RAM locations 0 to 9. A pulse occurs when the Stack Pointer is at 0 (one entry on Stack), and a Read-Stack Operation occurs to empty the Stack. A pulse also occurs when the Stack Pointer is equal to 7 (eight entries on the Stack), and a Write-Stack Operation occurs to fill the ninth word and leave one word empty so it may be used by the interrupt. When a Stack Interrupt occurs, the condition of the Stack can be determined by using the Stack-full Jump Condition (STFL); if the interrupt was due to a stack-full condition, STFL equals 1; if due to a stack-empty condition, STFL equals 0.

With the interrupt scheme described, an interrupt does not occur at initialize but does occur every time the Stack becomes empty. If the Stack is to be extended into memory, a Stack-empty Interrupt is required but may be inhibited by turning off IEN1 in other cases. (Refer to the PACE Assembly Language Programming Manual for examples of techniques for extending the stack into memory.) In order to prevent a Stack Interrupt when both hardware and software stacks become empty, a dummy word may be pushed on the Stack by the Initialize Routine.

If a Stack Interrupt occurs while there is a level-3 or a level-4 interrupt present and enabled, the stack interrupt pointer will be accessed incorrectly from location 0 instead of location 2. Therefore, if the stack interrupt is used in conjunction with level-3 or level-4 interrupts, the contents of location 0 must equal the contents of location 2, which contains the address of the user's stack interrupt service routine. Since location 0 (zero) is also the initialize address, this means that location 0 must contain a value that serves a dual purpose:

1. It serves as an instruction during initialization.
2. It serves as an address if a Stack Interrupt occurs at the same time as a level-3 or a level-4 interrupt.

For example, a Copy Flags to Register 0 Instruction (CFR) has an opcode of 0400. Thus, if this instruction were contained at location 0, it could serve as both the initialize instruction and as a pointer to the Stack Interrupt Service routine, which would begin at location 0400. A few precautions must be observed when using this technique:

1. Test and branch, Copy Register to Flags (CRF), and skip instructions should not be used in location 0.
2. JMP and JSR Instructions must be used with caution.
3. Location 1 must contain a jump to the user's initialize routine (unless location 0 contained a jump).
4. Instructions used for location 0 must have an opcode of X'0400 or greater since any lesser value will be interpreted as a Halt Instruction.

2.8 NHALT AND CONTIN SIGNALS

The NHALT Signal performs three different functions: programmed halt indicator output, processor stall input, and nonmaskable Level-0 Interrupt input. The CONTIN Signal is used as an interrupt acknowledge output signal, as an input signal to continue processor activity after a programmed halt or a processor stall, and in conjunction with NHALT to initiate the Level-0 Interrupt. The use of NHALT and CONTIN to accomplish these functions is described in the paragraphs that follow.

NOTE

> The CONTIN signal may also be used independently of the NHALT signal as a jump-condition input. Refer to 2.6 for details.

2.8.1 Programmed Halt

During normal program execution, the NHALT control line provides a high output. If a Halt Instruction is executed, the microprocessor NHALT output is driven low to indicate that microprocessor activity has been suspended. While PACE operation is in suspension, the NHALT output has a 7/8 duty cycle; that is, every eighth clock phase, the NHALT output goes high. The NHALT 7/8 cycle must be accounted for if the output is used as a logic signal but is of little concern if the output drives only a halt indicator. The NHALT output goes high after the Halt Instruction is terminated by application of the CONTIN Signal. The CONTIN input must go true (high) for a minimum of 16 clock cycles, and then low for 4 clock cycles for PACE operation to resume.

2.82 Processor Stall

To suspend operation of PACE under external control, the NHALT signal may be driven low by an external gate, overriding the NHALT output buffer internal to PACE. Microprocessor operation then is suspended after execution of the current instruction. The suspension may last for an indefinite period of time without loss of CPU status and may be terminated by use of the CONTIN input (properly sequenced with removal of the NHALT input). The timing sequence for the NHALT and CONTIN Signals is shown in figure 2-11. The NHALT and CONTIN method for suspending PACE operation can be useful for Direct Memory Access block data transfers which require full bus-throughput capacity.

2.8.3 Level-0 Interrupts

The Level-0 Interrupt is not maskable under program control and, therefore, is useful for alarm conditions (such as a power failure) or for implementing a software-based control panel. The PACE NHALT and CONTIN Signals are used to generate a Level-0 Interrupt. The required relative timing for Level-0 Interrupt generation is illustrated in figure 2-12, and, as shown, the CONTIN Signal can be used as an interrupt acknowledge to indicate that the interrupt is being processed by PACE.

For cases where an interrupt acknowledge is not required or where the CONTIN Signal is used as a sense input to the program, the CONTIN Signal can be held low continuously. While holding the CONTIN Signal continuously low, the NHALT Signal must be driven low at least for the duration of the longest instruction execution time plus eleven clock cycles to guarantee that a Level-0 Interrupt occurs.

When the NHALT Signal is subsequently driven to a high state, the Level-0 Interrupt servicing is initiated internally. Servicing consists of first setting the Level-0 Interrupt Enable (IR0 INT ENABLE in figure 2-10) low to lock out all other possible interrupts. Next, the contents of the PACE Program Counter are stored in the location specified by the contents of memory location 7 (see table 2-4). Then, the instruction at memory location 8 is executed. Storing the contents of the Program Counter in a memory location instead of on the Stack prevents generation of a Stack-full Interrupt.

NOTE

If a Level-0 Interrupt occurs within the 12-clock-cycle period (excluding extend cycles) following the recognition (indicated by CONTIN signal) of any other interrupt, the processor either will stall or execute the level-0 interrupt using the wrong pointer address. This problem may be avoided by only allowing the level-0 interrupt leading edge to be applied to the PACE chip during an NADS, *provided* no interrupt acknowledge has occurred since the last NADS. Figure 2-13 shows one circuit that can be used to accomplish this. Note that the circuit has been designed to take care of proper 'level-0' execution only. If one desires to 'STALL' also, proper control gating will have to be added to the circuit.

To return from a Level-0 Interrupt, the PFLG15 or SFLG15 Instruction is executed to set the Level-0 Interrupt Enable Output high after execution of one additional instruction. The additional instruction is typically a JMP@ through the memory location to which the contents of memory location 7 point; memory

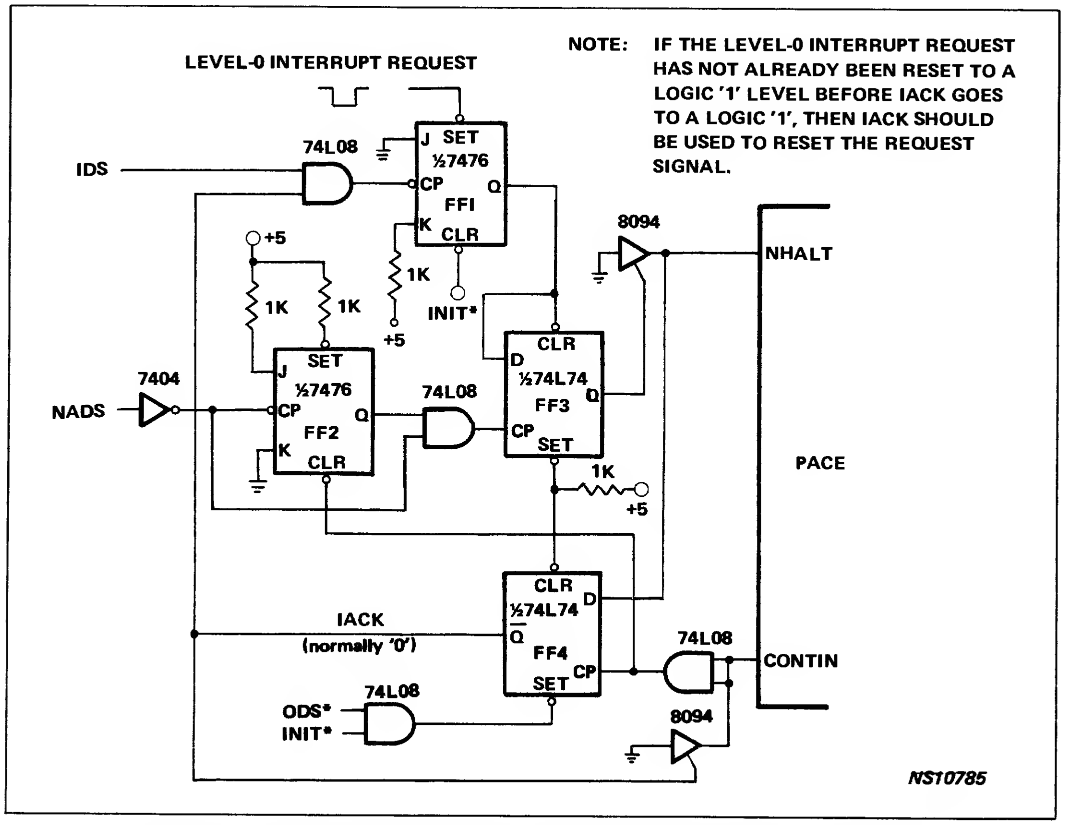

LEVEL-0 INTERRUPT REQUEST
NOTE: IF THE LEVEL-0 INTERRUPT REQUEST HAS NOT ALREADY BEEN RESET TO A LOGIC '1' LEVEL BEFORE IACK GOES TO A LOGIC '1', THEN IACK SHOULD BE USED TO RESET THE REQUEST SIGNAL.
IDS
74L08
J SET ½7476 CP Q FFI K CLR
+5
1K
1K
1K
INIT*
+5
8094
NHALT
7404
NADS
J SET ½7476 CP Q FF2 K CLR
74L08
CLR D ½74L74 Q FF3 CP SET
1K
+5
PACE
CLR D ½74L74 Q FF4 CP SET
IACK
(normally '0')
74L08
CONTIN
74L08
ODS*
INIT*
8094
NS10785

Figure 2-13. Circuit To Prevent Conflicts Between Level-0 and In-Process Interrupts

location 7 then restores the original contents of the Program Counter. Thus, a proper return to the interrupt program can be effected. For example, if memory location 7 contains X'1000, the PC contents are stored at memory location X'1000. To return from interrupt, a JMP@ X'1000 would be executed.

2.9 PACE INSTRUCTION SET

The PACE instruction set contains 45 instruction types that are capable of providing 337 individual instructions when flags, branch conditions, and other conditional signals or tests are considered. The 45 instruction types are divided into the following eight format groups:

- Branch Instructions
- Skip Instructions
- Memory Data-transfer Instructions (also serve as I/O instructions)
- Memory Data-operate Instructions
- Register Data-transfer Instructions
- Register Data-operate Instructions
- Shift and Rotate Instructions
- Miscellaneous Instructions

A summary of the PACE instructions is provided in table 2-5, which shows the instruction mnemonic, name, a symbolic representation of the instruction operation, and the instruction format. A more-detailed presentation of the instruction set is provided in appendix A.

There are no special PACE instructions for peripheral input/output. Instead, all of the memory-reference instructions can also be used with peripheral devices: this method provides a much wider variety of instructions for communications with peripherals.

2.9.1 Data Representation

In the PACE microprocessor, data are represented in the twos-complement number system, in which the negative of a number is formed by complementing each bit and, then, adding one to the complemented value of the number. The most significant bit indicates the sign of the number: 0 for positive and 1 for negative. With a single 16-bit value, the greatest positive number is X'7FFF or 32767_{10} and the most negative number is X'8000 or 32768_{10}. When the 8-bit data length is selected, the largest positive number is X'7F or 127_{10} and the most negative number is X'80 or 128_{10}.

2.9.2 Addressing Modes

Part of the power of the PACE microprocessor instruction set is derived from a flexible method of addressing used in memory-reference instructions (also used for peripheral devices). This method makes it possible to reference directly three 256-word 'pages' that may be located anywhere in memory, as well as another 256-word page in a fixed position in memory.

The fixed words form a 'base' page, and the others form three 'floating' pages. The mode of addressing is specified by the 2-bit XR field (bits 8 and 9) of the 16-bit instruction word, as shown in figure 2-14. The four available modes (base page, program-counter relative, AC2 relative, and AC3 relative) are summarized in table 2-6 and are described in the section that follows.

Table 2-5. PACE Instruction Summary

	Mnemonic	Name	Operation	Maximum Execution Time (Note)	Instruction Format
1.	**Branch Instructions**				
	BOC	Branch On Condition (Table 2-3)	(PC) ← (PC) + disp if cc true	$5M + E_R$ + 1M if branch	0 1 0 0 cc disp
	JMP	Jump	(PC) ← EA	$4M + E_R$	0 0 0 1 1 0 xr
	JMP@	Jump Indirect	(PC) ← (EA)	$4M + 2E_R$	1 0 0 1 1 0
	JSR	Jump To Subroutine	(STK) ← (PC), (PC) ← EA	$5M + E_R$	0 0 0 1 0 1
	JSR@	Jump To Subroutine Indirect	(STK) ← (PC), (PC) ← (EA)	$5M + 2E_R$	1 0 0 1 0 1
	RTS	Return from Subroutine	(PC) ← (STK) + disp	$5M + E_R$	1 0 0 0 0 0 0 0
	RTI	Return from Interrupt	(PC) ← (STK) + disp, IEN = 1	$6M + E_R$	0 1 1 1 1 1 0 0
2.	**Skip Instructions**				
	SKNE	Skip if Not Equal	If (ACr) ≠ (EA), (PC) ← (PC) + 1	$5M + 2E_R$ + 1M if skip	1 1 1 1 r xr disp
	SKG	Skip if Greater	If (AC0) > (EA), (PC) ← (PC) + 1	$7M + 2E_R$ + 1M if skip	1 0 0 1 1 1
	SKAZ	Skip if And is Zero	If [(AC0) ∧ (EA)] = 0, (PC) ← (PC) + 1	$5M + 2E_R$ + 1M if skip	1 0 1 1 1 0
	ISZ	Increment and Skip if Zero	(EA) ← (EA) + 1, if (EA) = 0, (PC) ← (PC) + 1	$7M + 2E_R + E_W$ + 1M if skip	1 0 0 0 1 1
	DSZ	Decrement and Skip if Zero	(EA) ← (EA) – 1, if (EA) = 0, (PC) ← (PC) + 1	$7M + 2E_R + E_W$ + 1M if skip	1 0 1 0 1 1
	AISZ	Add Immediate, Skip if Zero	(ACr) ← (ACr) + disp, if (ACr) = 0, (PC) ← (PC) + 1	$5M + E_R$ + 1M if skip	0 1 1 1 1 0 r
3.	**Memory Data Transfer Instructions**				
	LD	Load	(ACr) ← (EA)	$4M + 2E_R$	1 1 0 0 r xr disp
	LD@	Load Indirect	(AC0) ← ((EA))	$5M + 3E_R$	1 0 1 0 0 0
	ST	Store	(EA) ← (ACr)	$4M + E_R + E_W$	1 1 0 1 r
	ST@	Store Indirect	((EA)) ← (AC0)	$4M + 2E_R + E_W$	1 0 1 1 0 0
	LSEX	Load With Sign Extended	(AC0) ← (EA) bit 7 extended	$4M + 2E_R$	1 0 1 1 1 1
4.	**Memory Data Operate Instructions**				
	AND	And	(AC0) ← (AC0) ∧ (EA)	$4M + 2E_R$	1 0 1 0 1 0 xr disp
	OR	Or	(AC0) ← (AC0) ∨ (EA)	$4M + 2E_R$	1 0 1 0 0 1
	ADD	Add	(ACr) ← (ACr) + (EA), OV, CY	$4M + 2E_R$	1 1 1 0 r
	SUBB	Subtract with Borrow	(AC0) ← (AC0) + ~ (EA) + (CY), OV, CY	$4M + 2E_R$	1 0 0 1 0 0
	DECA	Decimal Add	(AC0) ← (AC0) $+_{10}$ (EA) $+_{10}$ (CY), OV, CY	$7M + 2E_R$	1 0 0 0 1 0
5.	**Register Data Transfer Instructions**				
	LI	Load Immediate	(ACr) ← disp	$4M + E_R$	0 1 0 1 0 0 r disp
	RCPY	Register Copy	(ACdr) ← (ACsr)	$4M + E_R$	0 1 0 1 1 1 dr sr 000 000
	RXCH	Register Exchange	(ACdr) ← (ACsr), (ACsr) ← (ACdr)	$6M + E_R$	0 1 1 0 1 1 dr sr
	XCHRS	Exchange Register and Stack	(STK) ← (ACr), (ACr) ← (STK)	$6M + E_R$	0 0 0 1 1 1 r 0 0
	CFR	Copy Flags Into Register	(ACr) ← (FR)	$4M + E_R$	0 0 0 0 0 1
	CRF	Copy Register Into Flags	(FR) ← (ACr)	$4M + E_R$	0 0 0 0 1 0
	PUSH	Push Register Onto Stack	(STK) ← (ACr)	$4M + E_R$	0 1 1 0 0 0
	PULL	Pull Stack Into Register	(ACr) ← (STK)	$4M + E_R$	0 1 1 0 0 1
	PUSHF	Push Flags Onto Stack	(STK) ← (FR)	$4M + E_R$	0 0 0 0 1 1 0 0
	PULLF	Pull Stack Into Flags	(FR) ← (STK)	$4M + E_R$	0 0 0 1 0 0 0 0

Table 2-5. PACE Instruction Summary (Concluded).

Mnemonic	Name	Operation	Maximum Execution Time (Note)	Instruction Format
6. Register Data Operate Instructions				
RADD	Register Add	(ACdr) ← (ACdr) + (ACsr), OV, CY	4M + E_R	0 1 1 0 1 0 · dr · sr · 000 000
RADC	Register Add With Carry	(ACdr) ← (ACdr) + (ACsr) + (CY), OV, CY	4M + E_R	0 1 1 1 0 1 · ↓ · ↓ · ↓
RAND	Register And	(ACdr) ← (ACdr) ∧ (ACsr)	4M + E_R	0 1 0 1 0 1 · ↓ · ↓ · ↓
RXOR	Register Exclusive OR	(ACdr) ← (ACdr) ∀ (ACsr)	4M + E_R	0 1 0 1 1 0 · ↓ · ↓ · ↓
CAI	Complement and Add Immediate	(ACr) ← ~ (ACr) + disp	5M + E_R	0 1 1 1 0 0 · r · disp
7. Shift And Rotate Instructions				
SHL	Shift Left	(ACr) ← (ACr) shifted left n places, w/wo link	(5 + 3n) M + E_R, n = 1 – 127; 6M + E_R, n = 0	0 0 1 0 1 0 · r · n · ℓ
SHR	Shift Right	(ACr) ← (ACr) shifted right n places, w/wo link		0 0 1 0 1 1 · ↓ · ↓ · ↓
ROL	Rotate Left	(ACr) ← (ACr) rotated left n places, w/wo link		0 0 1 0 0 0 · ↓ · ↓ · ↓
ROR	Rotate Right	(ACr) ← (ACr) rotated right n places, w/wo link		0 0 1 0 0 1 · ↓ · ↓ · ↓
8. Miscellaneous Instructions				
HALT	Halt	Halt		0 0 0 0 0 0 · 00 0 · 0000000
SFLG	Set Flag (Table 2-2)	$(FR)_{fc}$ ← 1	5M + E_R	0 0 1 1 · fc · 1 · ↓
PFLG	Pulse Flag (Table 2-2)	$(FR)_{fc}$ ← 1, $(FR)_{fc}$ ← 0	6M + E_R	0 0 1 1 · fc · 0 · ↓

Note: M = Machine cycle time = 4 clock periods
n = number of shifts
E_R = Extend time for read cycle
E_W = Extend time for write cycle
External interrupt response time is 7M + E_R plus time to finish current instruction.

Number	Mnemonic	Condition
0	STFL	Stack full
1	REQ0	(AC0) equal to zero(1)
2	PSIGN	(AC0) has positive sign(2)
3	BIT 0	Bit 0 of AC0 true
4	BIT 1	Bit 1 of AC0 true
5	NREQ0	(AC0) is non-zero(1)
6	BIT 2	Bit 2 AC0 is true
7	CONTIN	CONTIN (continue) input is true
8	LINK	LINK is true
9	IEN	IEN is true
10	CARRY	CARRY is true
11	NSIGN	(AC0) has negative sign(2)
12	OVF	OVF is true
13	JC13	JC13 input is true
14	JC14	JC14 input is true
15	JC15	JC15 input is true

Note 1: If the selected data length is 8 bits, only bits 0-7 of AC0 are tested.
Note 2: Bit 7 is the sign bit (instead of bit 15) if the selected data length is 8 bits.

Register Bit	Flag Name	Function
0	"1"	Not used—always logic 1
1	IE1	Interrupt Enable Level 1
2	IE2	Interrupt Enable Level 2
3	IE3	Interrupt Enable Level 3
4	IE4	Interrupt Enable Level 4
5	IE5	Interrupt Enable Level 5
6	OVF	Overflow
7	CRY	Carry
8	LINK	Link
9	IEN	Master Interrupt Enable
10	BYTE	8-bit data length
11	F11	Flag 11
12	F12	Flag 12
13	F13	Flag 13
14	F14	Flag 14
15	"1"	Always logic 1, set for Interrupt 0 exit

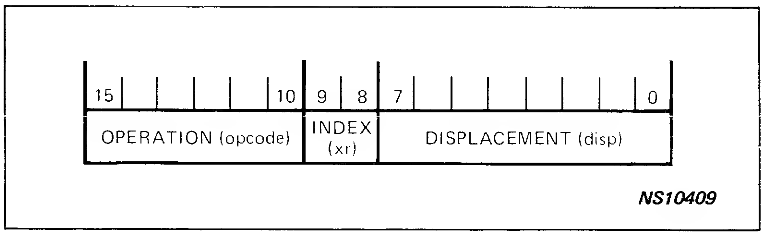


Figure 2-14. Memory-reference Instruction Format

Table 2-6. Summary of Direct Addressing Modes

xr Field	Addressing Mode	Effective Address
00	Base-page	EA = disp
01	Program-Counter-relative	EA = disp + (PC)
10	AC2-relative (indexed)	EA = disp + (AC2)
11	AC3-relative (indexed)	EA = disp + (AC3)

NOTES:
1. For base-page addressing, disp is positive and in range of 000 to 255 when BPS is low (0); or disp is signed number in range of -128 to +127 when BPS is high (1).
2. PC contains value one greater than address of current instruction.
3. For relative addressing, disp range is -128 to +127.

The PACE instruction set includes both direct and indirect memory addressing instructions. Both methods of memory addressing can use all of the addressing modes. The section that follows describes the operation of each of the available addressing modes as used in direct memory addressing. Indirect addressing is described in 2.9.2.2.

2.9.2.1 Direct Addressing

When the XR field is 00, it specifies base-page addressing. The base page may consist of either the first 256 words in the memory, or the first 128 plus last 128 words. The Base-Page-Select Signal (BPS) designates the option that will be used.

To address the first 256 words of memory (locations 0-255), BPS is set to 0; the 16-bit memory address is formed by setting bits 8 through 15 to zero and by using bits 0 through 7 to specify one of 256 locations.

If BPS is 1, the 16-bit memory address is formed by setting bits 8 through 15 equal to bit 7 and by using bits 0 through 6 to locate the first 128 words (X'0000-X'007F) of the memory (when bit 7 is 0) and the last 128 words (X'FF80-X'FFFF) (when bit 7 is 1). This technique is useful for splitting the base page between read-write and read-only memories or between memory and peripheral devices, so convenient base-page addressing can access data or peripherals.

When the XR field is 01, it specifies that addressing is relative to the Program Counter. In this mode, the memory address is formed by using bits 0 through 7 of the instruction word as a twos-complement displacement, with bit 7 being the sign bit. These bits form the less significant byte of a 16-bit displacement value, and the more significant byte is formed by propagating bit 7 (the sign bit) through bit 15. The displacement value thus formed then is added to the contents of the Program Counter, and the resulting sum thereby becomes the effective memory address. This addition is performed after the Program Counter has been incremented by 1 and, thus, is pointing to the next consecutive instruction address. Therefore, locations ranging from 128 locations below to 127 locations above the pre-incremented value of the Program Counter may be addressed using this technique.

When the XR field is 10 or 11, addressing is relative to an index register, and any memory location within the external 65,536-word address space may be referenced. As before, the displacement field is interpreted as a signed value ranging from -128 through +127. The memory address is then formed by adding the displacement bits to the contents of either Accumulator AC2 (when XR = 10) or Accumulator AC3 (when XR = 11). This type of addressing is desirable for those applications that require addresses to be computed at execution time, since addresses can not be modified when a ROM is serving for program storage (as is usually the case with microprocessors as opposed to minicomputers).

2.9.2.2 Indirect Addressing

Indirect addressing consists of first establishing an address in the same manner as direct addressing (by either the base-page, PC-relative, or indexed mode). The contents of the memory location at the selected address then are used as the operand address.

Chapter 3

DESIGNING PACE SYSTEMS: BASIC REQUIREMENTS AND CONCEPTS

3.0 INTRODUCTION

This chapter describes the basic elements required to obtain an operational PACE system and the components available to meet these requirements. System concepts are also discussed to provide designers with examples of system configurations and standard techniques that may be used in the design of PACE systems.

Figure 3-1 is a block diagram of an example of a PACE system configuration. The system illustrated includes the following:

- PACE – a detailed description of PACE is provided in chapter 2. Discussions of PACE in the remainder of this manual generally will be limited to explanations of how it is utilized in conjunction with other system elements.
- System Timing Element (STE) – this monolithic device provides the MOS clock signals required by PACE and also produces TTL clock signals for use by PACE and other system components. Use of the STE is described in 3.1.
- Address/Data and Control Signal Buffers – these Bidirectional Transceiver Elements (BTEs) are used to buffer the PACE MOS signals and to provide the MOS and TTL bus drive capability usually required in PACE systems. Details on buffering requirements and use of the BTEs are provided in 3.3.
- Address Latches – Requirements for address latches depend on the specific type of system bus structure used. Various system bus structures commonly used are discussed in 3.5 and implementation of address latches are discussed in 3.6.
- Memory – System memory requirements vary widely between applications. Chapter 4 of this manual is devoted to a discussion of memory selection criteria and interfacing considerations.
- Peripheral Interface – Although the PACE instruction set does not differentiate between memory and peripheral data transfers, there are a variety of specific capabilities provided by PACE that can be used to simplify peripheral interface designs. Chapter 5 provides a discussion of the capabilities available and the techniques for utilizing these capabilities.

The remainder of this chapter concentrates on what might be considered the "heart" of any PACE system–the nonshaded portion of figure 3-1–that is essentially independent of the system's particular application.

3.1 SYSTEM TIMING AND POWER

The timing requirements of PACE are fulfilled by the System Timing Element (STE) DP8302. The STE provides the true and complemented nonoverlapping clock inputs required by PACE and TTL clocks that can be used by other system components. The TTL clock output can also be used as described in 3.1.3 to simplify generation of the substrate voltage (V_{BB}) required by PACE.

The STE is contained in a single 16-pin dual-in-line package. A detailed description and electrical specifications for the STE are provided in the DP8302 data sheet. Figure 3-2 shows an example of an STE-to-PACE interconnection. The paragraphs that follow detail specific guidelines and techniques for using the STE in a PACE system.

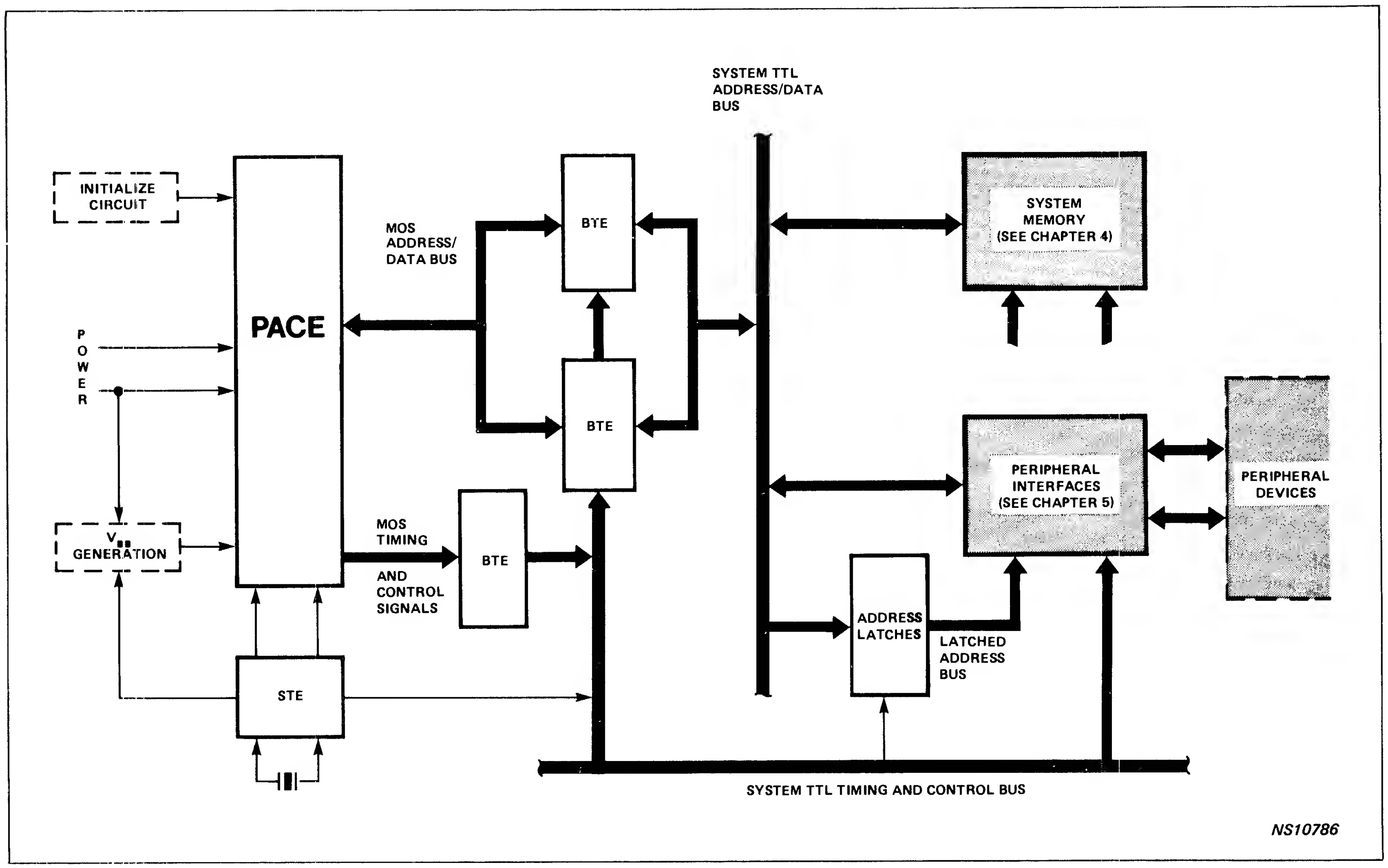

SYSTEM TTL ADDRESS/DATA BUS
INITIALIZE CIRCUIT
PACE
MOS ADDRESS/ DATA BUS
BTE
BTE
SYSTEM MEMORY (SEE CHAPTER 4)
POWER
PERIPHERAL INTERFACES (SEE CHAPTER 5)
PERIPHERAL DEVICES
VBB GENERATION
MOS TIMING AND CONTROL SIGNALS
BTE
ADDRESS LATCHES
LATCHED ADDRESS BUS
STE
SYSTEM TTL TIMING AND CONTROL BUS
NS10786

Figure 3-1. Example of PACE System Configuration

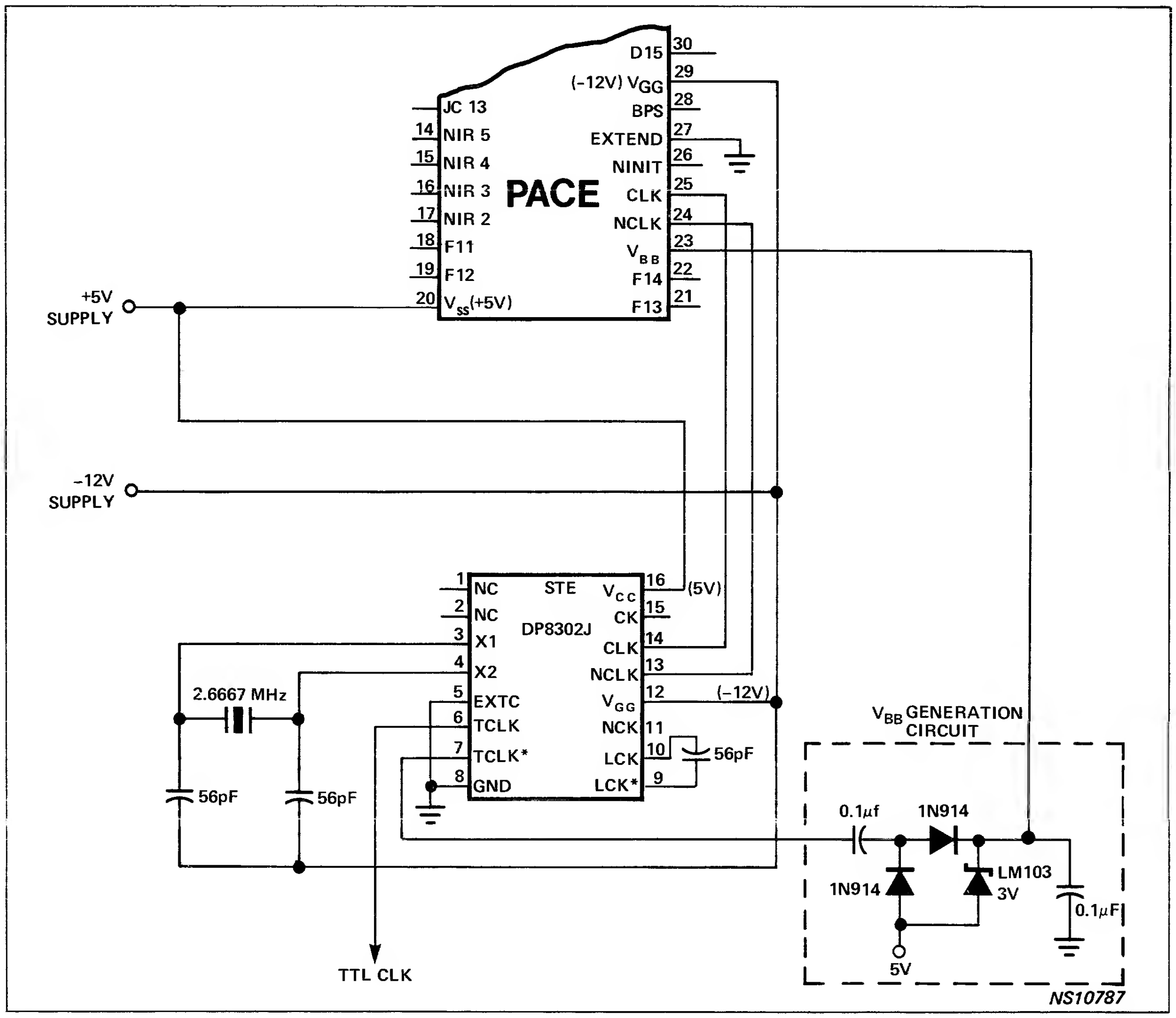


Figure 3-2. PACE-STE Interconnection and V_{BB} Generation

3.1.1 Frequency Control

An external series-resonant crystal is connected between pins X1 and X2 of the STE to provide frequency control. Alternately, an external TTL clock input may be applied to the STE, bypassing the internal oscillator. In this case, pin X1 must be tied to V_{GG} and pin X2 must be left open. Then, EXTC may be used as a TTL input for the external oscillator.

The output frequency of the STE is one half of the input frequency. Thus, a 2.6667-megahertz crystal connected to the STE results in clock inputs (CLK and NCLK) to PACE at a frequency of 1.3333-megahertz. This frequency is equivalent to a clock period of 750 nanoseconds – the optimal clock period for PACE (IPC-16A/520D).

3.1.2 Nonoverlap Requirements and Control

The clock nonoverlap requirements for PACE are specified in the IPC-16A/520D data sheet. The STE incorporates a cross-coupled latch containing a delay in the feedback path that ensures nonoverlapping MOS clock signals. The delay in the feedback path can be increased by connecting a capacitor between pins LCK and NLCK on the STE. The effect of the capacitor on increasing the nonoverlap interval is shown in the STE data sheet.

3.1.3 Substrate Bias Voltage (V_{BB}) Generation

Both PACE and the STE require +5-volt and −12-volt power. Additionally, PACE requires a substrate bias voltage (V_{BB}) of +8 volts. Figure 3-2 shows a circuit that uses one of the TTL clock outputs of the STE to derive the required voltage.

3.2 SYSTEM INITIALIZATION

The NINIT input to PACE should be used after the system has been powered-up and clock signals applied to PACE (refer to 2.3 for a detailed discussion of initialization). The NINIT signal can be reapplied at any time to reinitialize PACE.

Figure 3-3 shows a circuit that can be used to produce an NINIT signal of the required duration both on power-up and when initiated by a user-supplied switch. The output of the DM74132 Schmitt Trigger can also be used to initialize other system devices.

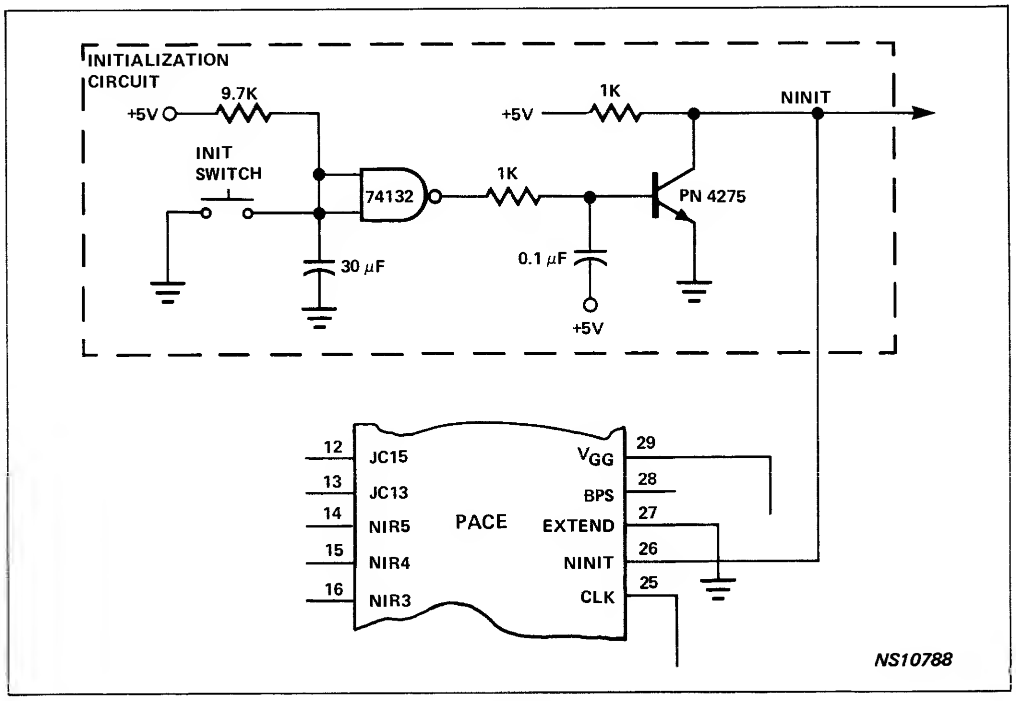


Figure 3-3. Initialization Circuit

3.3 BUFFERING SYSTEM BUSES

It is possible to design a limited PACE system that does not require buffering of PACE signals. However, most systems that require the powerful capabilities of PACE also include other devices in sufficient number to demand a high fanout of the PACE address/data lines and the PACE control signals. These demands can be satisfied easily using the Bidirectional Transceiver Elements (BTEs) DP8300N that have been specifically developed to simplify the design of PACE systems.

The BTE provides buffering between the PACE MOS input/output lines and TTL devices. A high-fanout capability of up to 30 TTL loads (50 milliamperes) is provided by the BTE (refer to the DP8300 data sheet for specifications). A functional block diagram of the BTE is shown in figure 3-4.

Figure 3-5 shows an example of the system implementation of the BTE. One BTE is connected to operate only in the drive mode by grounding the Write Bus Data pin (WBD*). No connections are made to the BTE Chip Enable inputs (CE and CE*) or Strobe Input (STR*) since, as indicated by the BTE truth table in figure 3-5, when the WBD* input is low the states of the other control signals are "don't care." Thus, the BTE operating in the drive-only mode provides buffering for the PACE MOS timing and control signal outputs.

Two BTEs are used in figure 3-5 to buffer the PACE bidirectional address/data lines. Directional control of the BTEs is implemented in a straightforward manner using the Buffered Input Data Strobe (BIDS) signal from PACE (via the "drive-only" BTE described in the preceding paragraph). BIDS is connected to the WBD* mode control input of both BTEs. The other three mode control inputs are continuously enabled by connecting CE1 to a logic '1' level, and CE2* and STR* to ground (logic '0'). Referring to the truth table in figure 3-5, it can be seen that the BTEs will be in the "drive TTL and receive MOS" mode when BIDS is low (logic 0). This allows address output and data output from PACE to be placed onto the System TTL Address/Data Bus without using any additional control or enable signals. When BIDS is high (logic 1), indicating that PACE is awaiting input data, the BTEs are placed in the "receive TTL and drive MOS" mode to allow data from the System TTL Address/Data Bus to be applied to PACE.

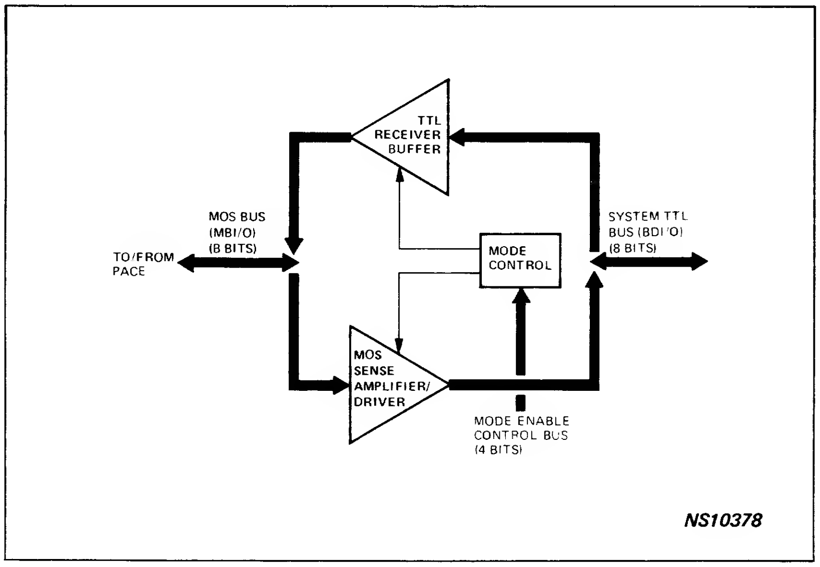
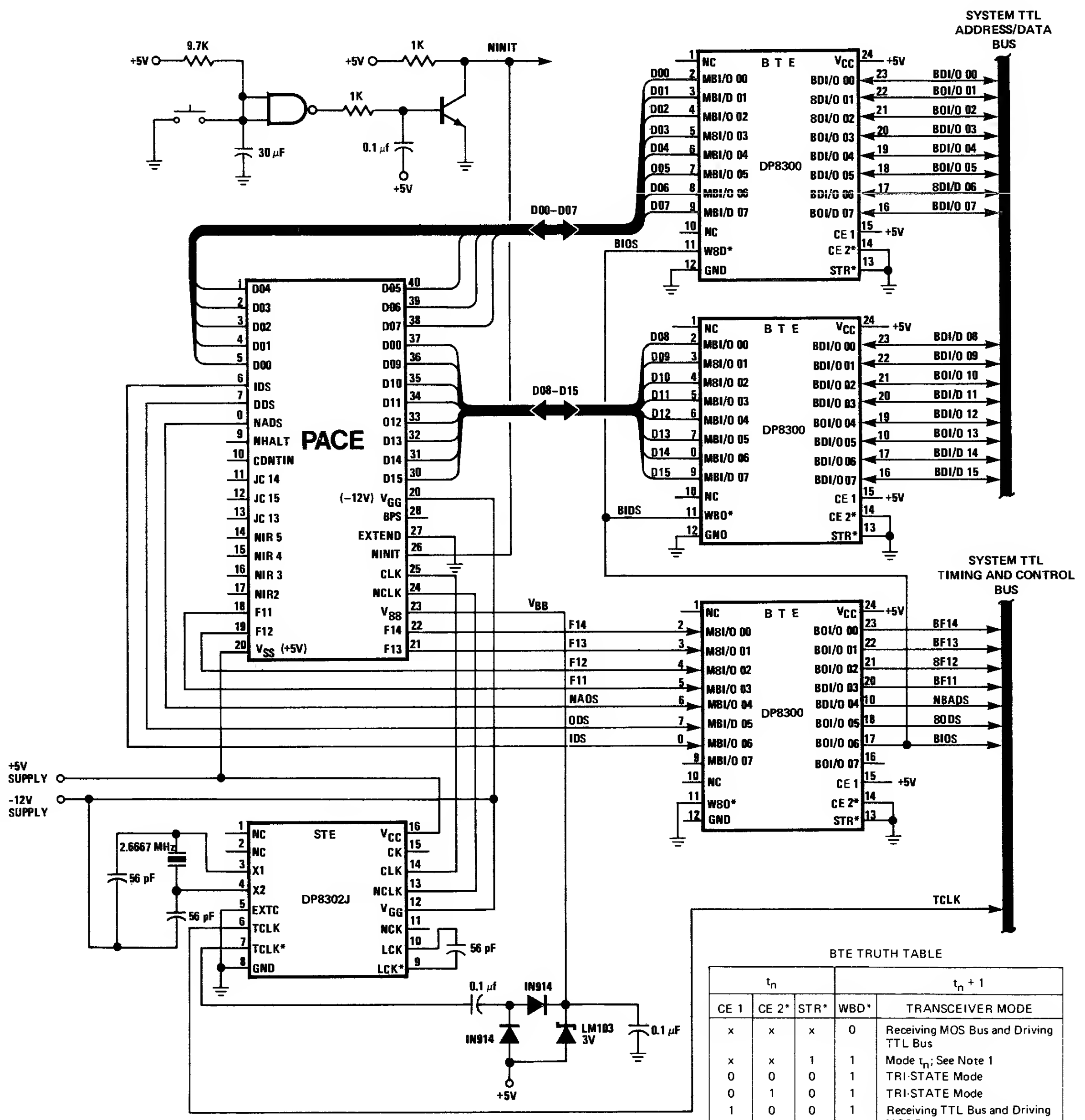

TTL RECEIVER BUFFER
MOS BUS (MBI/O) (8 BITS)
TO/FROM PACE
MODE CONTROL
SYSTEM TTL BUS (BDI/O) (8 BITS)
MOS SENSE AMPLIFIER/ DRIVER
MODE ENABLE CONTROL BUS (4 BITS)
NS10378

Figure 3-4. BTE Functional Block Diagram



BTE TRUTH TABLE

t_n			$t_n + 1$	
CE 1	CE 2*	STR*	WBD*	TRANSCEIVER MODE
x	x	x	0	Receiving MOS Bus and Driving TTL Bus
x	x	1	1	Mode t_n; See Note 1
0	0	0	1	TRI-STATE Mode
0	1	0	1	TRI-STATE Mode
1	0	0	1	Receiving TTL Bus and Driving MOS Bus
1	1	0	1	TRI-STATE Mode

Note 1. On the positive-edge transition of STR* logic conditions present on CE 1 and CE 2* at the time of transition will be latched internally. The transceiver will either be in the TRI-STATE or receiving mode.

Figure 3-5. BTE System Implementation

The control scheme just described provides simple directional control of the BTEs to easily implement a bi-directional System TTL Address/Data Bus. Other methods of course can be used if a particular system demands. For example, in a system that utilizes Direct Memory Access (DMA), it may be desirable to place the TTL outputs of the BTEs in the high-impedance (TRI-STATE®) mode when other system devices are transferring data on the bus.

3.4 A BUFFERED PACE CPU MODULE

Figure 3-6 is a block diagram representation of figure 3-5 – a fully buffered PACE CPU module – based on the devices, circuits, and interconnection examples described previously in this chapter. The module depicted in figure 3-7 provides a fully operational CPU that can be used as the basis for a variety of systems regardless of their application. In order to simplify the descriptions of system concepts and interfacing techniques presented in the remainder of this chapter and in chapters 4 and 5, the Buffered PACE CPU MODULE shown in figure 3-6 is referenced frequently.

The remainder of this chapter discusses system bus structures, address latching, and address decoding. While these areas are more application-dependent than the topics discussed thus far in chapter 3, the concepts apply equally to memory interfacing and peripheral interfacing and, therefore are discussed in this chapter as an adjunct to the CPU module.

3.5 SYSTEM BUS STRUCTURES

Methods of buffering the PACE address/data lines and timing and control signals to obtain a System TTL Address/Data Bus and System TTL Timing and Control Bus are discussed in 3.3. The resulting fully multiplexed, bidirectional address/data bus is suitable for many systems and minimizes wiring costs and device counts. Utilization of such a bus structure is simplified by the full complement of timing and control signals provided by PACE.

Figure 3-7 illustrates a system that uses this fully multiplexed bus structure. The memory devices in such a system would be of the type that include on-chip address latches. A peripheral interface could be implemented by providing an input/output port for peripheral data and address decoding logic to enable the input/output port. (Examples of address decoding are provided in 3.7.) A variety of other system bus structures may also be implemented to meet particular system demands.

In many systems, the separate-address/multiplexed-data bus structure depicted in figure 3-8 can be used to simplify interfaces to memory and peripheral devices. In systems where memory devices do not supply on-chip address latches, this bus structure provides a latched address that can be utilized by both system memory and by address comparator/decoders that might be necessary for peripheral interfaces. Implementation of address latches to obtain a separate address bus is described in 3.5.

A variety of other bus structures can be configured to meet particular system demands. For example, in systems where a large number of devices must be serviced, it might be advantageous to provide separate buses and buffering for memory and peripheral devices. More detailed examples of various bus structures are provided in chapter 4 (Memory Interfacing) and chapter 5 (Peripheral Interfacing).

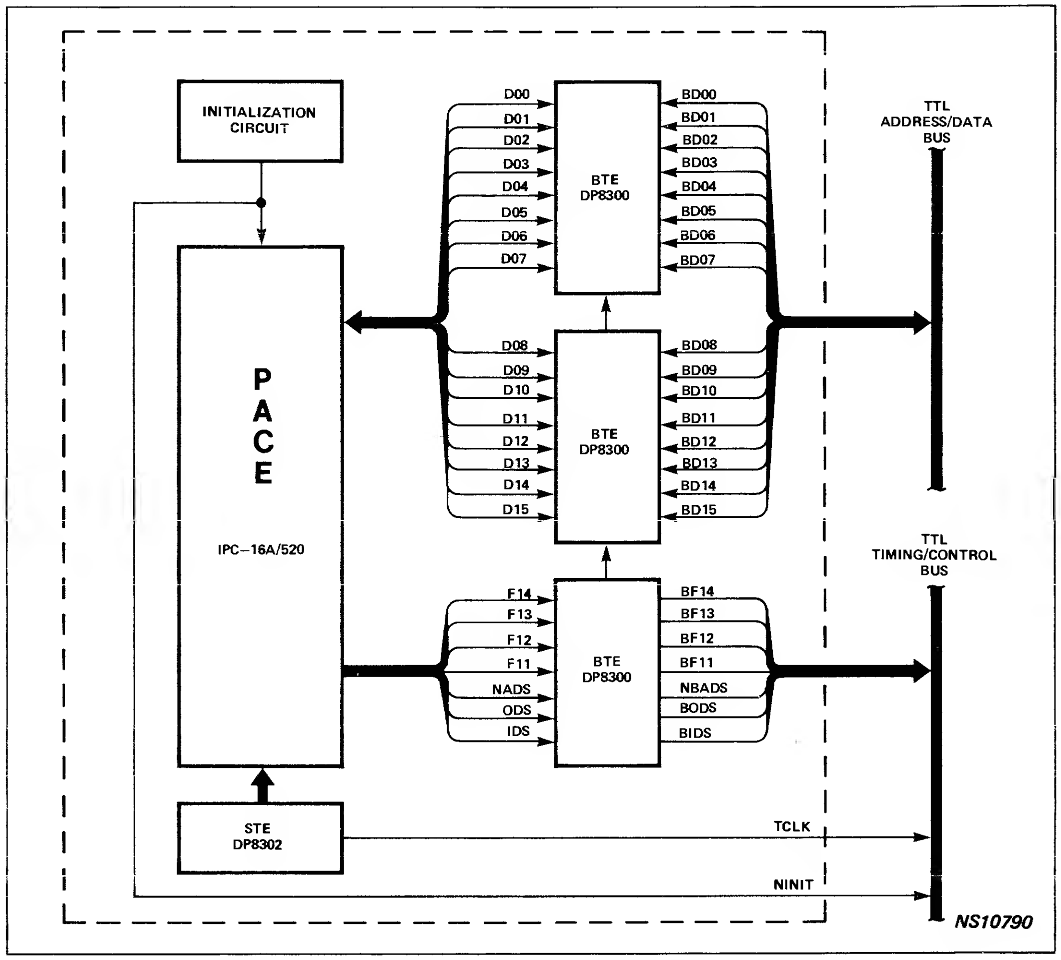


Figure 3-6. Buffered PACE CPU Module

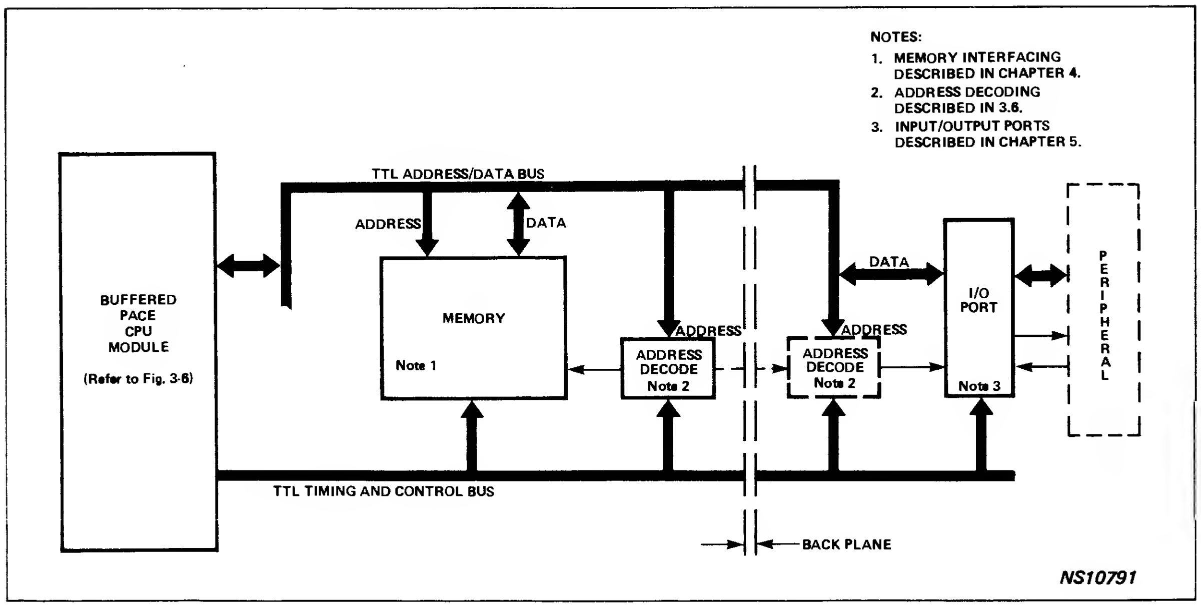


Figure 3-7. Fully Multiplexed Bus Structure

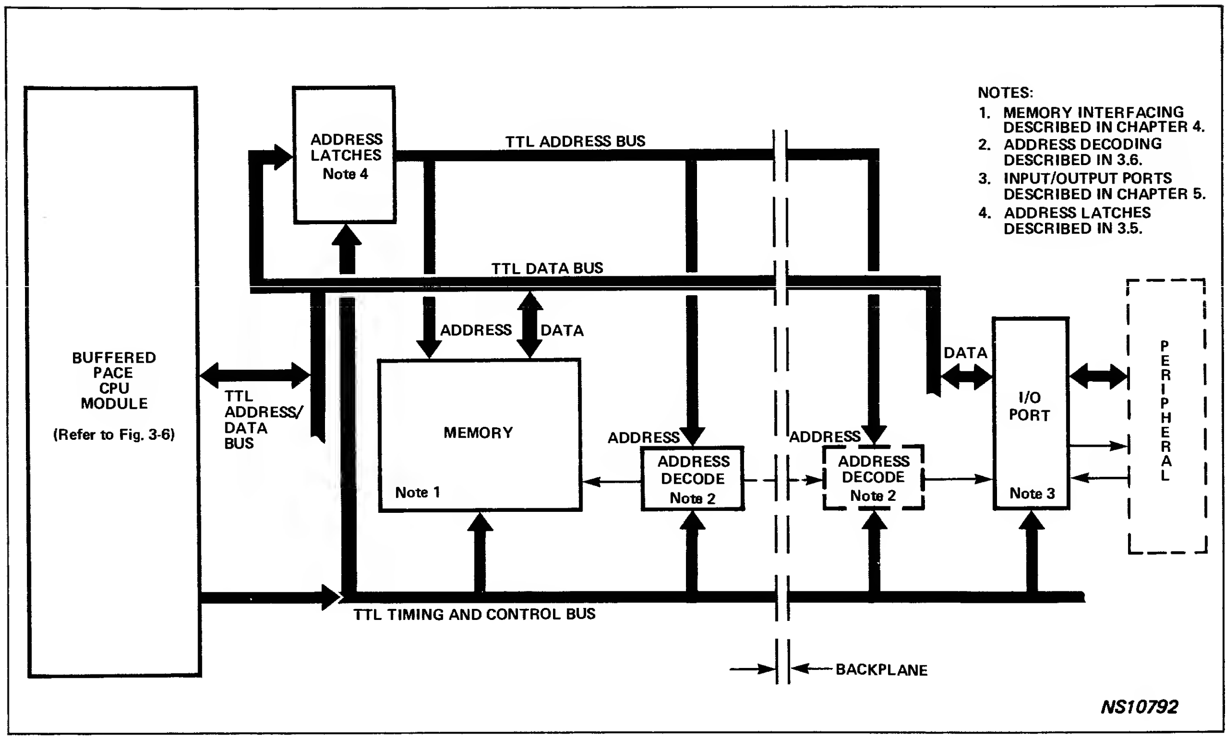


Figure 3-8. Separate Address/Multiplexed Data Bus Structure

3.6 ADDRESS LATCHING

In systems where it is desirable to have separate buses for output of addresses and input/output of data, address latches must be provided to capture the information from the multiplexed address/data bus. During the address-output interval of each data input/output operation, the negative-true Address Data Strobe (NADS) signal is output by PACE. NADS is active in the middle of valid address information. Thus, either edge of NADS can be used to clock address information into address latches.

Figure 3-9 illustrates one method of providing a latched address bus. Two DMLS174 Hex Flip-Flops and one DMLS175 Quad Flip-Flop are used to capture the 16 bits of buffered address information from the system TTL Address/Data Bus. The Buffered NADS signal (NBADS) is used as the CLOCK input to the three devices. The CLEAR input to the flip-flops is not used in the example but could be connected to a system initialization signal if a particular application requires.

3.7 ADDRESS DECODING

The 16-bit address word output by PACE during each data input/output operation provides a total address space of 65,536 (64K) locations. The less significant address bits are typically used directly as address inputs to memory devices. For example, a 256-word RAM device requires eight address inputs (address bits 0 through 7), while a 1024-word ROM device requires ten address inputs (address bits 0 through 9). The more significant address bits thus are available for use as chip-select or chip-enable inputs to memory devices or as device select signals for peripherals.

The number of address bits required to select or enable memory or peripheral devices is directly related to the amount of memory and the number of peripheral devices in a system. Since the same PACE instructions are used for memory references and peripherals, the system designer must allocate specific address spaces to peripherals and others for memory. Allocation of address spaces is accomplished by decoding selected address bits to derive the required memory-select or peripheral-select signals.

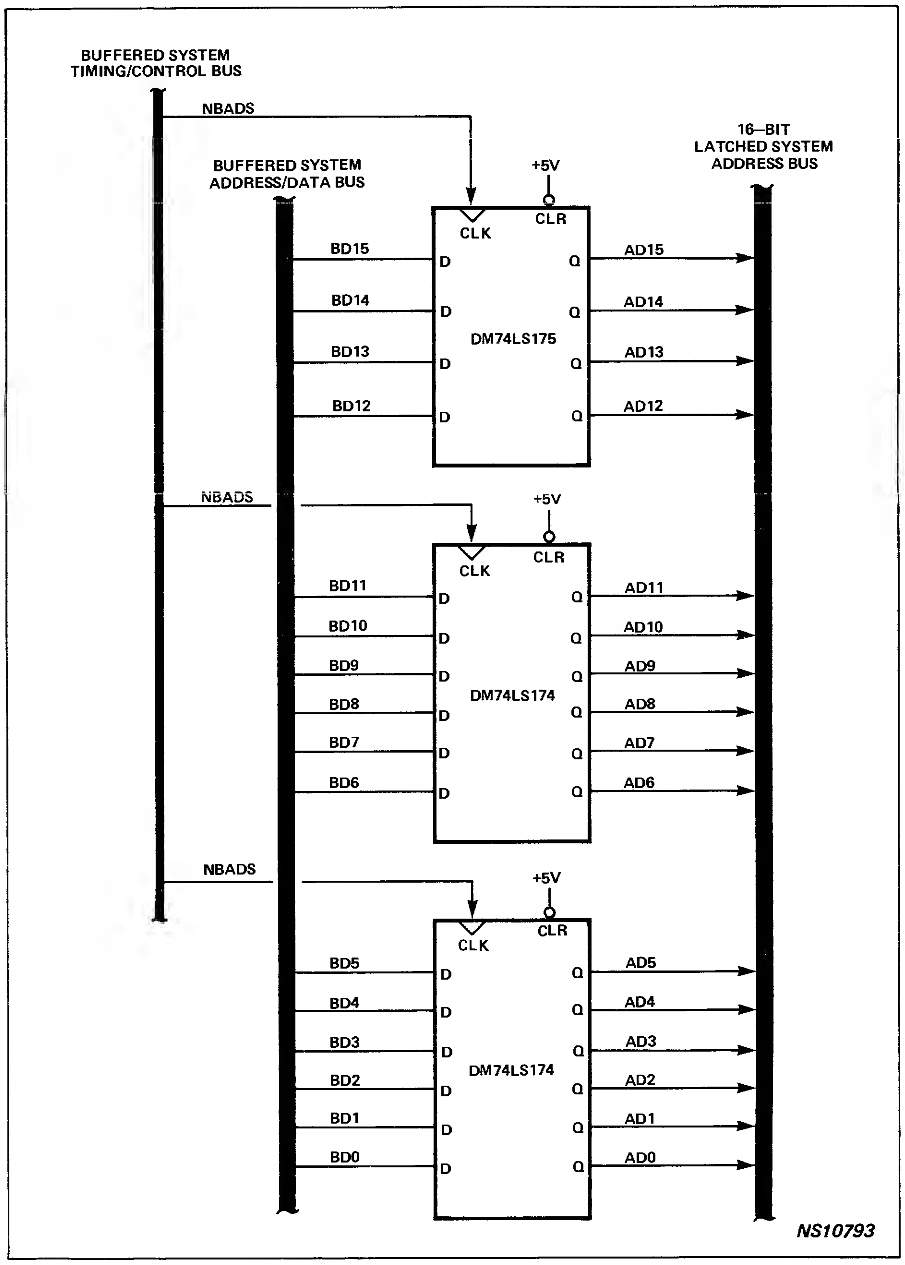


Figure 3-9. Latched Address Bus Using Hex/Quad Flip Flops

"Address decoding" may not always require the use of an actual decoding device or logic. For example, in a small system consisting of a limited amount of memory and a single peripheral device, the most significant address bit (bit 15) could be used to differentiate between memory and peripheral operations. With this method, half of the 64K address space would be allocated to memory and half to the peripheral: for example, when bit 15 is low (logic 0), the memory address space is active, and when bit 15 is high the peripheral device is selected. If additional memory and/or peripheral devices are present in the system, other address bits can be used directly as the exclusive select or enabling signals for these devices. Figure 3-10 shows an example of how individual address bits might be used to select six separate peripheral devices.

NOTE

Address bits that are not required for use as select signals can be used to transfer other information such as command or control codes to peripherals. Refer to 5.5 for a discussion of this technique.

Using address bits directly as select signals eliminates the need for extra decoders – an important consideration in small systems. This method, however, does limit the number of peripheral devices and memory that can be addressed because it makes inefficient use of the address space and because it may result in complicating system software. For example, with the scheme shown in figure 3-10, an address in the range X'C000-X'C1FF selects device #1 by setting bit 14 to a logic '1'; however, an address in the range X'FE00–X'FFFF sets bits 9–14 to a logic '1' and, thus, selects all six peripheral devices.

As system size increases, more extensive address decoding schemes may be necessary to achieve more efficient utilization of address space. A variety of decoding devices can be used, ranging in complexity from combinations of AND/OR gates through standard MSI decoders. Figure 3-11 shows a DM74LS138 3-to-8 line decoder being used to generate 8 peripheral select signals. Address bit 15 is used as the enable input and address bits 14, 13, and 12 as the binary select inputs to the decoder. As shown in the resulting address map, each of the peripheral devices occupies a separate 4K address space in the upper half of the 64K total available addresses.

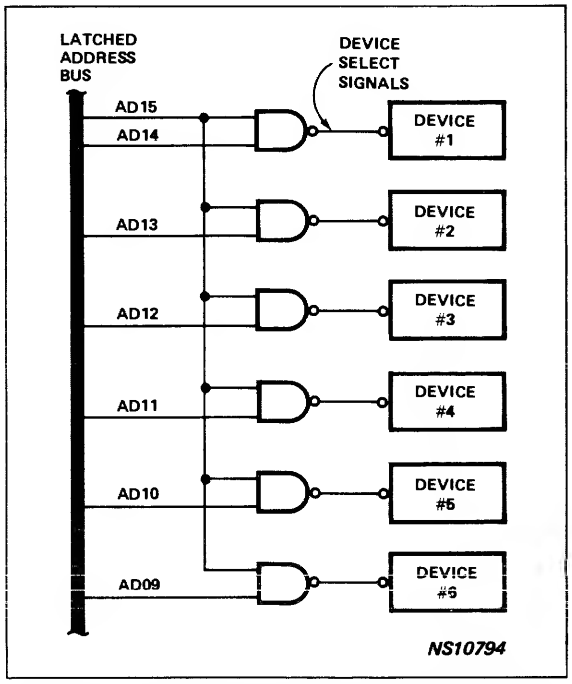
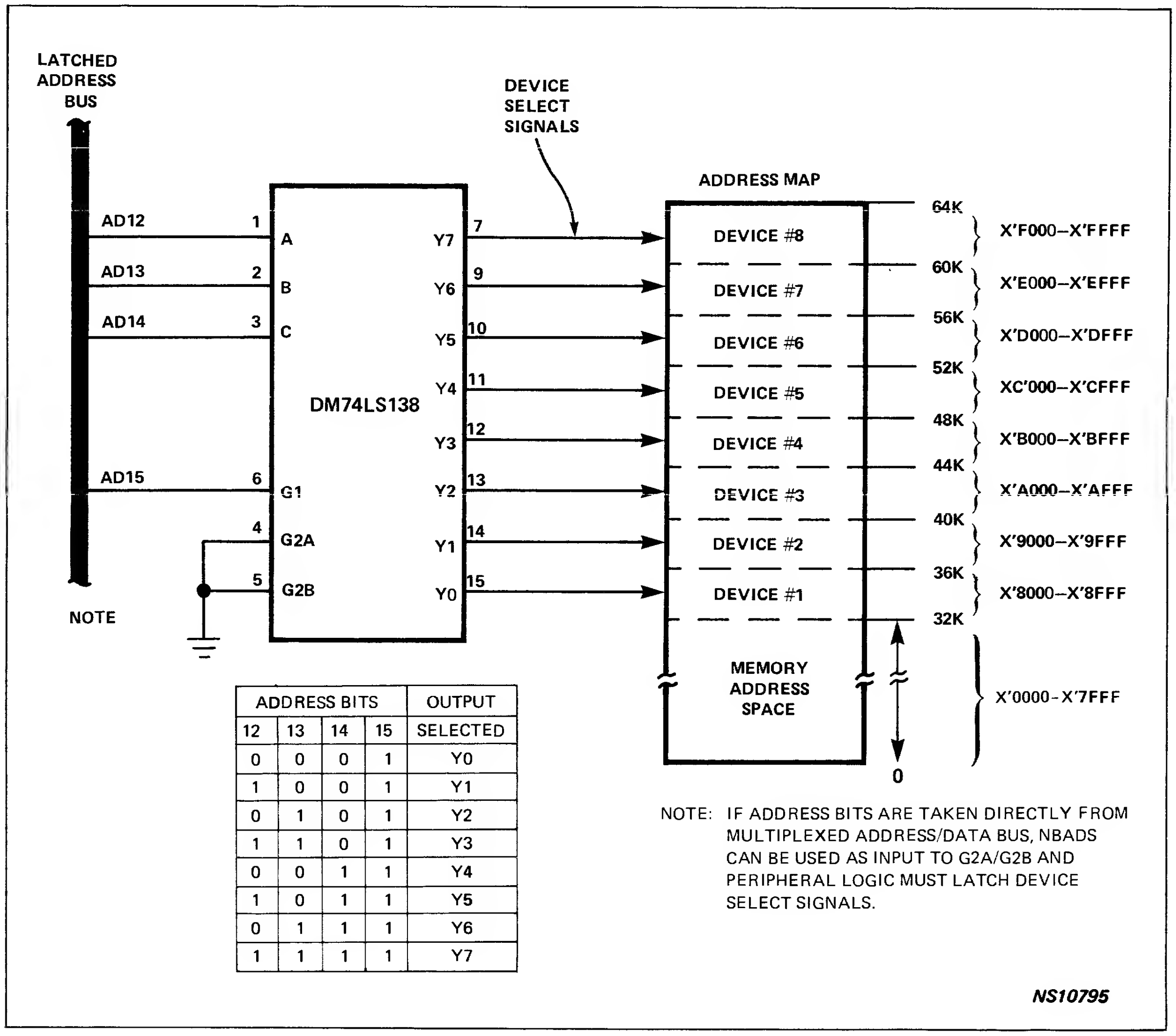


Figure 3-10. Using Address Bits as Device Select Signals

Another method of address decoding is shown in Figure 3-12. Here, a 6-bit unified-bus comparator (DM8131) is used to enable a 1-line to 8-line demultiplexer. The DM8131 compares the contents of the Address/Data Bus (at NADS time) against a user-designated hardwired address (the T1–T6 inputs). In the figure, the output of the comparator will go low (logic '0') whenever an address in the range X'3000–X'37FF is detected. The output of the comparator is combined (using a NOR gate) with NBADS to provide the required logic '0' level to the DATA input of the DM8223 demultiplexer. The address inputs (A, B, and C) to the DM8223 are latched address bits A08, A09, and A10, respectively. The address bits are latched using a DM74175 Quad D Flip-Flop. This decoding scheme provides negative-true enabling signals (such as might be required for memory devices) for eight 256-word blocks of addresses.



ADDRESS BITS				OUTPUT
12	13	14	15	SELECTED
0	0	0	1	Y0
1	0	0	1	Y1
0	1	0	1	Y2
1	1	0	1	Y3
0	0	1	1	Y4
1	0	1	1	Y5
0	1	1	1	Y6
1	1	1	1	Y7

NOTE: IF ADDRESS BITS ARE TAKEN DIRECTLY FROM MULTIPLEXED ADDRESS/DATA BUS, NBADS CAN BE USED AS INPUT TO G2A/G2B AND PERIPHERAL LOGIC MUST LATCH DEVICE SELECT SIGNALS.

Figure 3-11. Using a 3-to-8 Line Decoder to Generate Device Select Signals

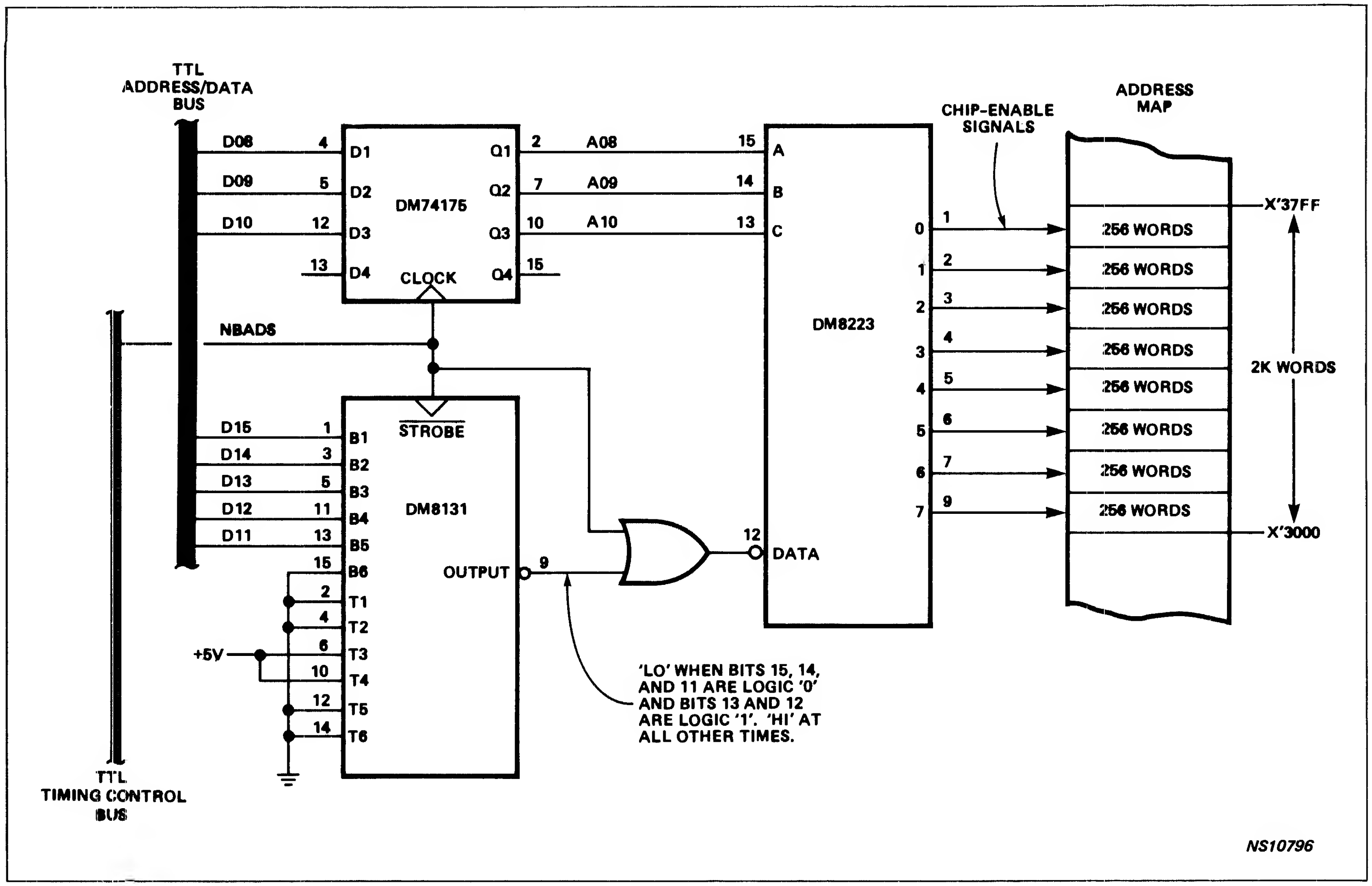


Figure 3-12. Using a Bus Comparator and a Demultiplexer to Enable Memory Devices

Chapter 4

MEMORY INTERFACING

4.0 INTRODUCTION

The amount and type of memory used in a PACE system is determined by a variety of application-dependent factors that include program size, data storage requirements, speed considerations, total system parts count, and system power constraints. Table 4-1 lists some standard memory devices available from National Semiconductor that can be used with PACE systems. The paragraphs that follow provide examples of how some of these memory devices can be utilized in PACE systems, and also describe general considerations that apply when interfacing PACE to memory.

Table 4-1. Standard Memory Devices for Use in PACE Systems

Memory Type	Part Number	Pins	Description
RAM	MM2101	22	256 x 4 static MOS RAM, 500-1000ns access
	MM2102	16	1K x 1 static MOS RAM, 500-1000ns access
	MM2112	16	256 x 4 static MOS RAM, 650-1000ns access
	MM5269	22	256 x 4 static MOS RAM, with address latches, 1000ns access
	MM5271	18	4K x 1 dynamic MOS RAM, 250ns access TTL compatible
	MM5281	22	4K x 1 dynamic MOS RAM, 250ns access TTL compatible
ROM	DM87S202	20	256 x 8 Schottky ROM, with output data latches, 90ns access
	MM5214	24	512 x 8 MOS ROM, 1000ns access
	MM5242	24	1K x 8 MOS ROM, 500ns access
	MM5246	24	2K x 8 MOS ROM, 500ns access
PROM	MM5204	24	512 x 8 electrically programmable MOS ROM, 1000ns access
	DM87S222	20	256 x 8 Bipolar PROM with output latches, 60ns access

4.1 INTERFACING TO MEMORY WITH ON-CHIP LATCHES

Figure 4-1 illustrates a typical PACE system implementation of memory using memory devices that provide on-chip latches. Use of these types of devices eliminates the need for supplying a separate system latched address bus and permits memory to be interfaced directly to the multiplexed System TTL Address/Data

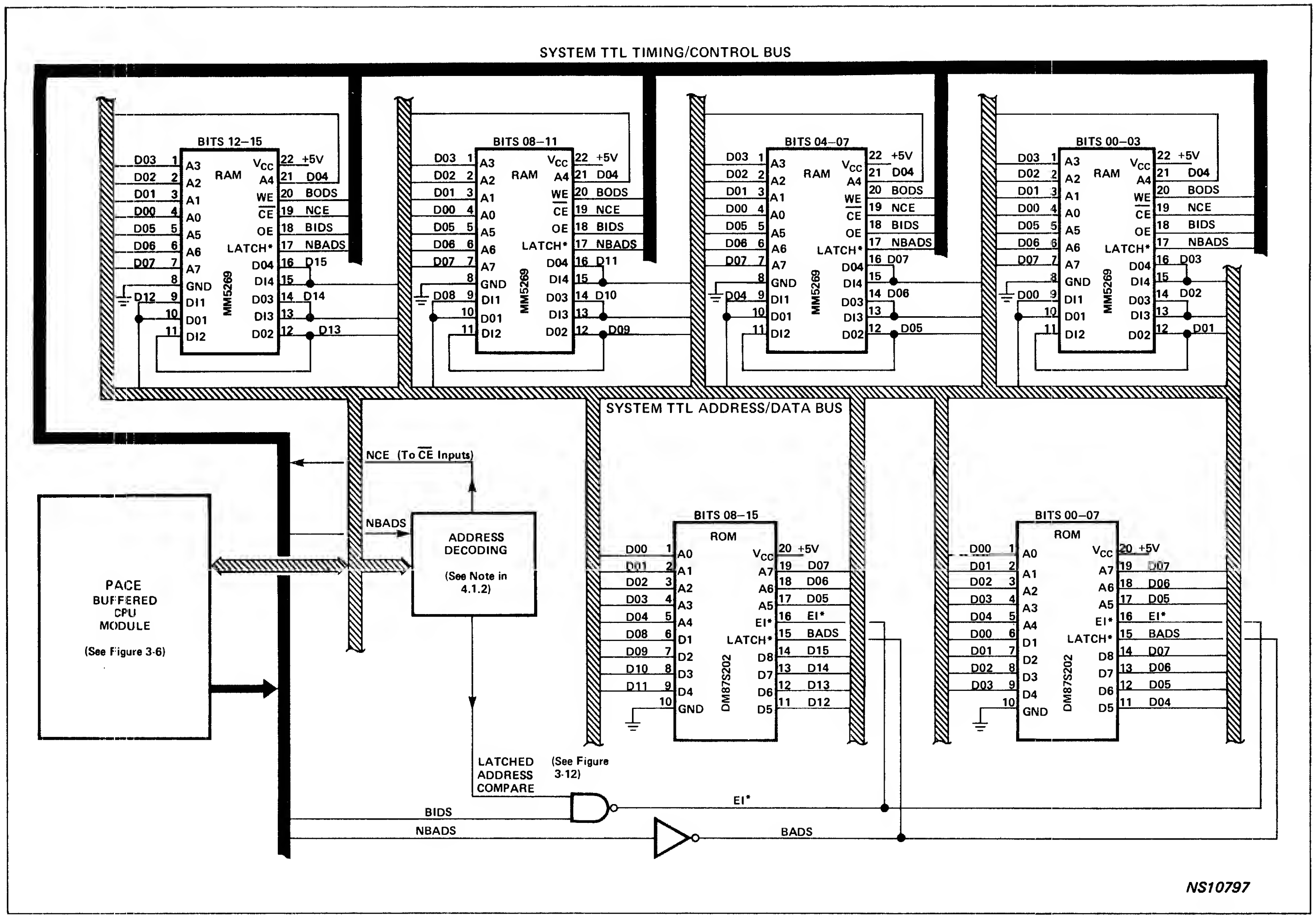

SYSTEM TTL TIMING/CONTROL BUS
SYSTEM TTL ADDRESS/DATA BUS
BITS 12–15
BITS 08–11
BITS 04–07
BITS 00–03
RAM
MM5269
BITS 08–15
BITS 00–07
ROM
DM87S202
PACE BUFFERED CPU MODULE (See Figure 3-6)
ADDRESS DECODING (See Note in 4.1.2)
NCE (To CE Inputs)
NBADS
LATCHED ADDRESS COMPARE (See Figure 3-12)
BIDS
EI*
BADS
NS10797

Figure 4-1. Typical Implementation of Memory With On-chip Latches

Bus. Since there are subtle differences in the ways that RAM and ROM are interfaced to PACE, each interface is discussed separately in the two sections that follow.

4.1.1 A Typical Interface to RAM With On-Chip Latches

The RAM devices in the upper portion of figure 4-1 are MM5269 static MOS RAMs which provide 256-by-4 bits of read/write memory. The MM5269 contains on-chip address latches and a chip-enable latch to simplify interfacing to the multiplexed address/data bus.

Four control inputs are provided on each device: CE* (Chip Enable), LATCH, WE (Write Enable), and OE (Output Enable). The LATCH Signal causes the inputs to the RAM address pins (AD00–AD07) to be latched into an internal address register on the RAM to select one of the 256 4-bit words within the RAM. At the same time the address bits are latched, the Chip Enable Input (CE*) is latched into an internal register to prepare the device for a memory read or memory write cycle. The memory read and write cycles are selected by the state of the Output Enable (OE) and Write Enable (WE) signals respectively.

The buffered PACE Address Data Strobe (NBADS) signal can be used as the LATCH input to RAM, and an address bit or an output from an address decoder can be used as the input to the Chip Enable (CE*) pin of each RAM. For example, to position RAM in address space X'FF00–X'FFFF, the address decoder depicted in figure 4-1 would produce the NCE signal when address bits 8 through 15 were all high (logic 1). Fewer bits could be used if system address space is not fully utilized; in a simple system it might be sufficient to merely use address bit 15 as the enabling signal. (A detailed discussion of address decoding is provided in section 3.7.)

The buffered PACE Input Data Strobe (BIDS) signal can be connected to the Output Enable (OE) input to the RAMs to initiate a memory read cycle when PACE is performing a data-input operation. Similarly, the buffered Output Data Strobe (BODS) signal from PACE can be used to initiate a memory write cycle by connecting BODS to the Write Enable (WE) input to the RAMs.

Each MM5269 device provides 256 4-bit words. Thus, four devices must be connected in parallel to obtain a 16-bit word. This is accomplished in figure 4-1 by connecting bits D0–D3 from the System TTL Address/Data Bus to the rightmost RAM, D4–D7 to the next RAM, and so on until D15 has been connected to the most significant data lines of the leftmost RAM. Note that these RAM devices provide separate pins for data input and data output lines. In figure 4-1 the input and output pins are merely connected together to the appropriate bit of the System TTL Address/Data Bus. This can be done because the data output lines from each RAM are placed in the high-impedance state except when data is actually being output. However, in some systems, where long signal lines are required or many devices are attached to the System TTL Address/Data Bus, it may be necessary to provide additional output buffering.

4.1.2 A Typical Interface to ROM With On-Chip Latches

The read-only-memory (ROM) devices shown in figure 4-1 are DM87S202 bipolar ROMs that are organized in a 256 word by 8-bit configuration. The on-chip latches provided by these devices are output data latches instead of address latches as were supplied by RAM in the preceding section. The high-speed of this ROM allows data to be accessed and presented to the ROM output data latches within the time interval that the PACE NADS signal is true. The data can then be gated out onto the address/data bus under control of the PACE IDS signal. Thus, the on-chip data latches have the same effect on interfacing as the address latches provided by RAM – they eliminate the need for providing a separate system latched-address bus.

The two control inputs to the DM87S202 are LATCH* and EI*. The NBADS signal from the PACE Buffered CPU Module is inverted (becoming BADS) and connected to LATCH*. When BADS is true (high), the contents of the addressed memory word fall through the ROM data latches to an output buffer. When BADS goes low, the data is latched and the address inputs may be changed without affecting the latched data.

The output buffers are controlled by the EI* signal. While the EI* signal is high, the buffer outputs are in a TRI-STATE® (high-impedance mode). When EI* goes low, the contents of the data latches are gated out onto the system bus. In figure 4-1, the BIDS (Buffered Input Data Strobe) Signal is ANDed with a decoded address signal (ADDRESS COMPARE) to generate the EI* signal. Thus, the selected word will be placed on the System TTL Address/Data Bus when PACE is expecting input data.

Bits 0 through 7 from the Address/Data Bus are connected to the address inputs of the ROM devices. Since no chip select input is required, the devices will continuously be addressed, and will access and latch data each time the BADS Signal is applied to LATCH*. The data will not be gated onto the system address/data bus, however, unless EI* is generated by the combination of BIDS and a decoded address signal.

NOTE

The address decoding mechanism required with these ROM devices is slightly different from what is necessary with the MM5269 RAMs discussed in the preceding section. The Chip Enable (CE*) Input to RAM is latched internally by the RAM and thus the decoded address signal need not be latched and could simply consist of an address bit or an output from an AND gate. With the DM87S202 devices, on the other hand, the decoded address signal must be present at the same time as BIDS. Therefore, the address decoder used must provide a latched output. For a detailed discussion of various address decoding methods, refer to section 3.7.

4.2 INTERFACING TO MEMORY WITHOUT ON-CHIP LATCHES

Figure 4-2 illustrates a typical PACE system implementation of memory using devices that do not provide on-chip latches. When devices of this type are used, the system must include a latched address bus to provide stable presentation of address information to the memory devices throughout memory read/write operations.

Implementation of a 16-bit latched address bus is described in 3.6. In some systems, however, it may not be necessary to provide a full 16-bit address bus. For example, in the system depicted in figure 4-2. the maximum number of bits required to address a memory device is 9 (A0–A8 for the MM5214 ROM). Thus, two hex latches could be used to provide a 12-bit address bus: bits 0–7 would be used to address RAM; bits 0–8 would be used to address ROM; and three bits (for example, 9, 10, and 11 or 9, 10, and 15) would be available for address decoding to generate the chip-select signals for the memory devices and other system components. (Refer to 3.7 for a detailed discussion of address decoding techniques.)

Once the latched address bus is provided for the memory devices in figure 4-2, the remainder of the PACE/ memory interface is straightforward. The sections that follow (4.2.1 and 4.2.2) briefly describe the interfaces to the memory devices shown in figure 4-2; these interfaces are typical and the methods described apply generally to other memory devices without on-chip latches.

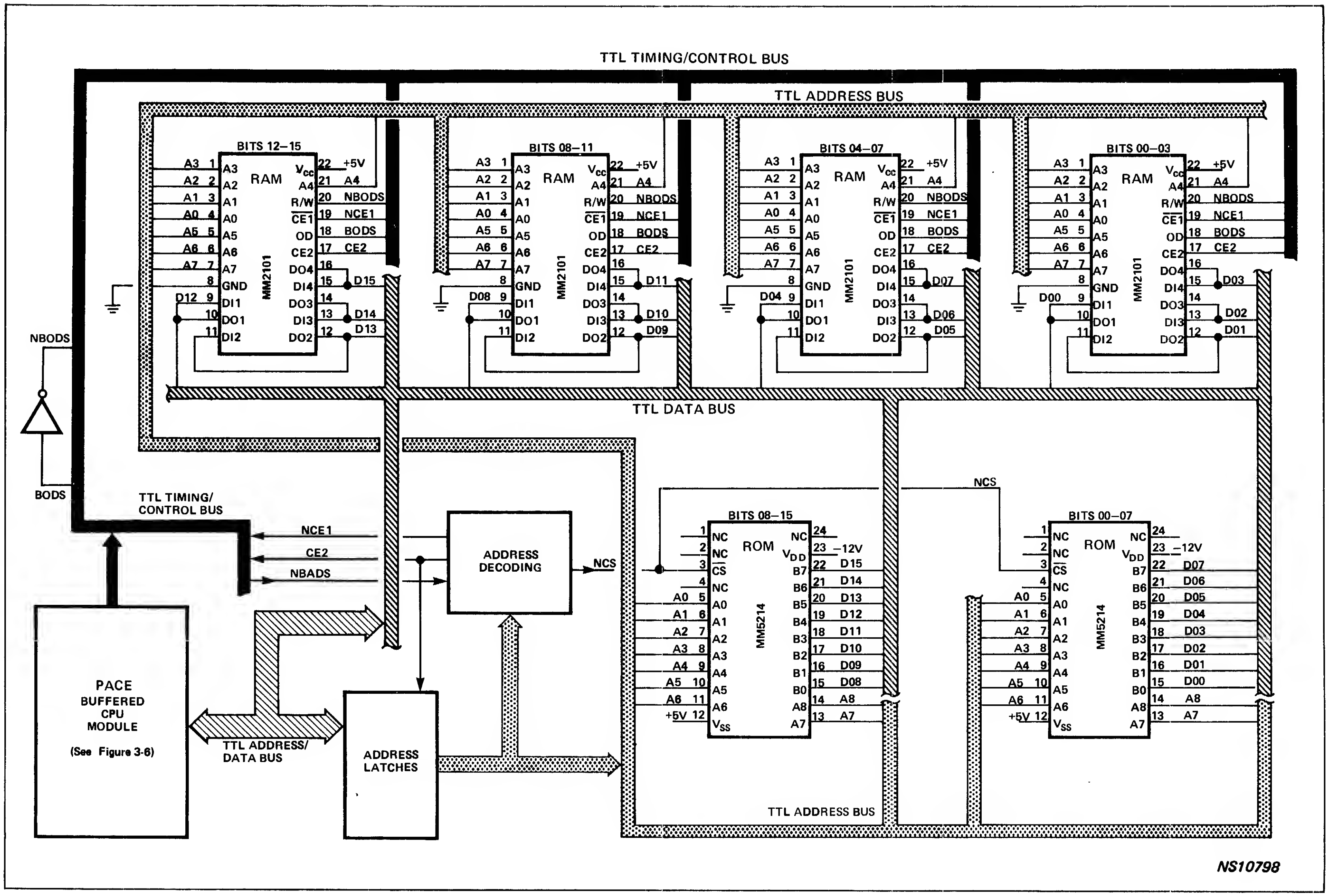


Figure 4-2. Typical Implementation of Memory Without On-chip Latches

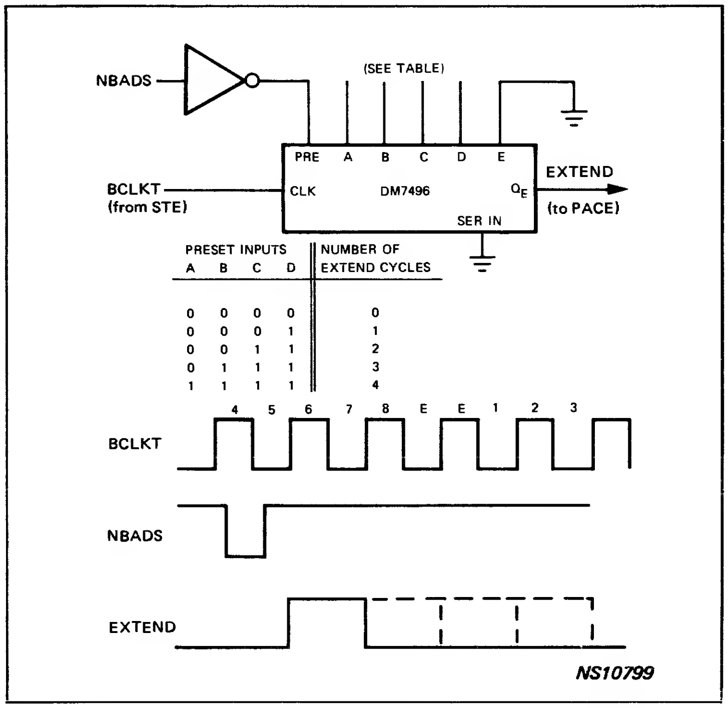
4.2.1 Interfacing to MM2101 RAM

The MM2101 Random Access Memory (RAM) devices provide 256-by-4 bits of read/write data storage. Four control inputs (R/W, $\overline{CE1}$, CE2, and OD) are used to control operation of the RAM. The $\overline{CE1}$ and CE2 inputs are chip enable inputs and are generated by the address decoding logic (NCE1 and CE2 respectively) in figure 4-2. The R/W Signal must be a high (logic 1) for read operations and low (logic 0) for write operations. In figure 4-2, the Buffered Output Data Strobe (BODS) signal is inverted (becoming NBODS) and used as the R/W input. This technique supplies a high to R/W during read operations, and a low-going pulse in the center of valid data during write operations.

The OD signal controls the data output lines from RAM: when OD is high the DO_0-DO_3 output data lines are in the high-impedance mode; when OD is low the contents of the addressed memory word are gated out onto the system data bus. In figure 4-2, the PACE Buffered Output Data Strobe (BODS) signal is used as the OD input. BODS is high during write (PACE data output) operations and thus causes the RAM data output lines to be placed in the high-impedance mode. This allows the RAM data input and data output pins to be connected together to the system data bus. BODS is low at all other times and thus allows data addressed during read operations to be gated out to the system data bus.

4.2.2 Interfacing to MM5214 ROM

The MM5214 ROM provides 512-by-8 bits or read-only-memory. The Chip Select ($\overline{CS}$) input is the only control signal required with this device and, in figure 4-2, $\overline{CS}$ is generated by a signal (NCS) from the address decoding logic. The contents of the addressed word are gated onto the system data bus and then are captured by PACE during a data input operation.



PRESET INPUTS A	B	C	D	NUMBER OF EXTEND CYCLES
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	1	1	1	3
1	1	1	1	4

Figure 4-3. Circuit and Timing Diagram for Up to Four Clock Cycle Extend

4.3 INTERFACING WITH SLOW MEMORIES

With system timing based on a crystal frequency of 2.667 megahertz, the resulting PACE clock period is 750 nanoseconds. A PACE data input/output operation allows approximately two clock periods for data access; thus, no special timing considerations are required if memory access time is less than approximately 1.5 microseconds.

All of the memory devices discussed thus far in this chapter have access times in the range of 1 microsecond or less. If slower memory devices are used or if system bus loading factors make timing marginal, the EXTEND input to PACE can be used to lengthen the input/output cycle to accommodate the increased access time.

A detailed description of the EXTEND signal and the required timing relationships during input/output operations are provided in 2.4.3. A circuit that can be used to generate the EXTEND signal within the required parameters is shown in figure 4-3.

NOTE

If the EXTEND signal is not used, it should be tied to ground.

The EXTEND signal increases the input/output cycle time by multiples of the clock period. In figure 4-4 a DM7496 5-bit Shift Register is used to provide an extend of up to four clock cycles. The number of extra cycles is determined by the level on the preset inputs, as shown in the accompanying table.

The circuit in figure 4-3 would result in an EXTEND during every input/output cycle. In some systems, for example a system that contains a mix of slow and fast memory, it may not be necessary to extend every input/output cycle. In such a system, it may be advantageous to perform address decoding so that the preset (PRE) input to the DM7496 shown in figure 4-3 is generated only when slow memory is addressed. Thus, the EXTEND signal would be generated only when required and the system could operate at maximum speed with other devices.

Chapter 5

PERIPHERAL INTERFACING TECHNIQUES

5.0 INTRODUCTION

While, as mentioned in preceding chapters, PACE instructions do not differentiate between memory and peripheral devices, the characteristics of peripheral device interfaces and the methods of transferring information between PACE and peripheral devices are much more varied than those associated with memory. Among the common methods used for peripheral data transfers are:

- 16-bit and/or 8-bit parallel data transfers
- Bit-serial data transfers
- Program-controlled data transfers
- Interrupt-driven or -initiated data transfers
- Direct-memory-access data transfers

In addition to the wide variety of methods commonly utilized to transfer data between central processing units and peripherals, special controls of enabling signals are frequently required to implement efficient peripheral interfaces. This chapter describes techniques for utilizing the resources of PACE to assist in the design of efficient interfaces for a variety of peripheral devices.

5.1 PERIPHERAL INSTRUCTIONS AND ADDRESSING

Data transfers between PACE and other system devices occur during each instruction access and during data accesses required by memory-reference instructions. Since all data transfers, whether with memory or peripheral devices, occur through execution of memory-reference instructions, the class of instructions used by PACE to effect data transfers could properly be called the input/output reference class. In contrast, many central processing units have one instruction type (input/output class) for communicating with peripherals and another type (memory-reference class) for communicating with memory. The approach used by PACE is more powerful because a wider variety of instructions (the entire memory-reference class) is available for communications with peripherals. Thus, for example, the Decrement and Skip if Zero Instruction (DSZ) can be used to decrement a peripheral device register, or the Skip if AND is Zero Instruction (SKAZ) can be used to test the contents of a peripheral device status register. For simple data transfers, the Load (LD) and Store (ST) Instructions can be used to accomplish the transfer of 16-bit or 8-bit parallel data words.

No special addressing considerations must be observed with peripheral data transfers: the user must merely ensure that specific address spaces are allocated for peripherals as described in 3.6. All of the addressing modes available for memory-reference instructions (see 2.9.2) can also be used with peripherals.

5.2 INPUT/OUTPUT PORTS

The physical characteristics of peripheral devices and their relatively slow response time typically require that data that is to be input from or output to peripherals be captured by an intermediate system element. Such elements, usually called input/output ports, thus hold the data to be transferred until either PACE or the peripheral device has utilized the data.

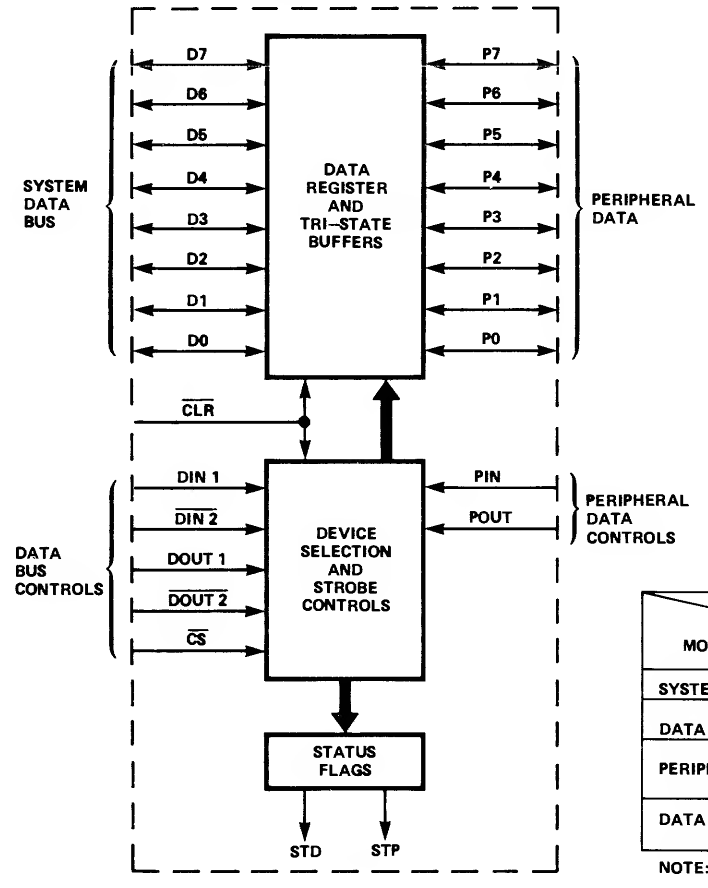
A specially designed PACE support chip, the Microprocessor Interface Latch Element (MILE) is available to simplify implementation of input/output ports. The MILE (DP8301) is a bidirectional 8-bit latch with TRI-STATE® output buffers, device selection logic, and status flags for "handshake" control or interrupt generation. Figure 5-1 illustrates the internal organization of the MILE.

The Device Selection and Strobe Control Logic allows the MILE to be used as either a bidirectional or unidirectional data latch. The truth table in figure 5-1 summarizes the modes that are available. Note that for each data input/output pin, input controls override the output controls but that input and output of opposite sides may occur simultaneously. In addition, both outputs (D0-D7 and P0-P7) may be active simultaneously, but not both inputs. In typical operation, however, only one mode is enabled at a time. Refer to the DP8301 data sheet for detailed electrical and timing specifications for the MILE.

The MILE supplies one status bit for each direction of transfer to indicate the status of the data in the Data Register. The STD status bit indicates that bus data is in the Data Register to be read by the peripheral. The STP status bit indicates that peripheral data is in the Data Register to be read by the system bus. These signals are active high and are reset by reading data out of the Data Register (from the opposite side). Thus, these status bits can be used for "handshake" input/output or for interrupt-driven input/output.

NOTE

Refer to 5.4 for additional details on "handshake" input/output, and 5.8 for a discussion of interrupt-driven input/output.



MILE TRUTH TABLE

MODE \ CONTROL SIGNALS →	DIN1	$\overline{\text{DIN2}}$	$\overline{\text{CS}}$	DOUT1	$\overline{\text{DOUT2}}$	$\overline{\text{PIN}}$	POUT
SYSTEM DATA BUS → DATA REGISTER	1	0	0	X	X	X	X
DATA REGISTER → SYSTEM DATA BUS	0	X	0	1	0	X	X
	X	1	0	1	0	X	X
PERIPHERAL DATA → DATA REGISTER	0	X	X	X	X	1	X
	X	1	X	X	X	1	X
	X	X	1	X	X	1	X
DATA REGISTER → PERIPHERAL DATA	X	X	X	X	X	0	1
	1	0	0	X	X	X	1

NOTE: X = "DON'T CARE"

NS10800

Figure 5-1. MILE Block Diagram

Figure 5-2 shows a typical system implementation of two MILEs as a bidirectional 16-bit peripheral input/output port and a single MILE as an 8-bit peripheral output port. PACE data are transferred to/from the MILE on the System TTL Address/Data Bus. The Chip Select ($\overline{CS}$) inputs to the MILEs must be low (logic 0) to enable transfer of data to and from MILE via the System TTL Address/Data bus. The address decoding logic depicted in figure 5-2 decodes the contents of the Address/Data Bus when NBADS is low and generates a chip select signal for either the upper two MILEs (NCS1) or the lower MILE (NCS2).

NOTE

The Chip Select Signals must remain low for the duration of the data transfer operation. The address decoding logic must therefore provide latched outputs so that NCS1 or NCS2 are valid when BIDS or BODS occurs. Refer to 3.7 for a detailed discussion of address decoding logic.

The BIDS and BODS signals from the System TTL Timing/Control Bus are used as inputs to DOUT1 and DIN1, respectively, to provide the required data directional control signals for the two MILEs used as bidirectional input/output ports.

The MILE at the lower righthand corner of figure 5-2 is shown connected for output data transfers only and BODS is used as the input to DIN1. For system operation with this 8-bit input/output port, user-generated software may use the Set Flag (SFLG) Instruction to set the PACE BYTE Flag. Thus, PACE ignores the high-order bits (D08–D15) and operates only on the low-order bits (D00–D07). The following section discusses system considerations when interfacing to 8-bit peripherals.

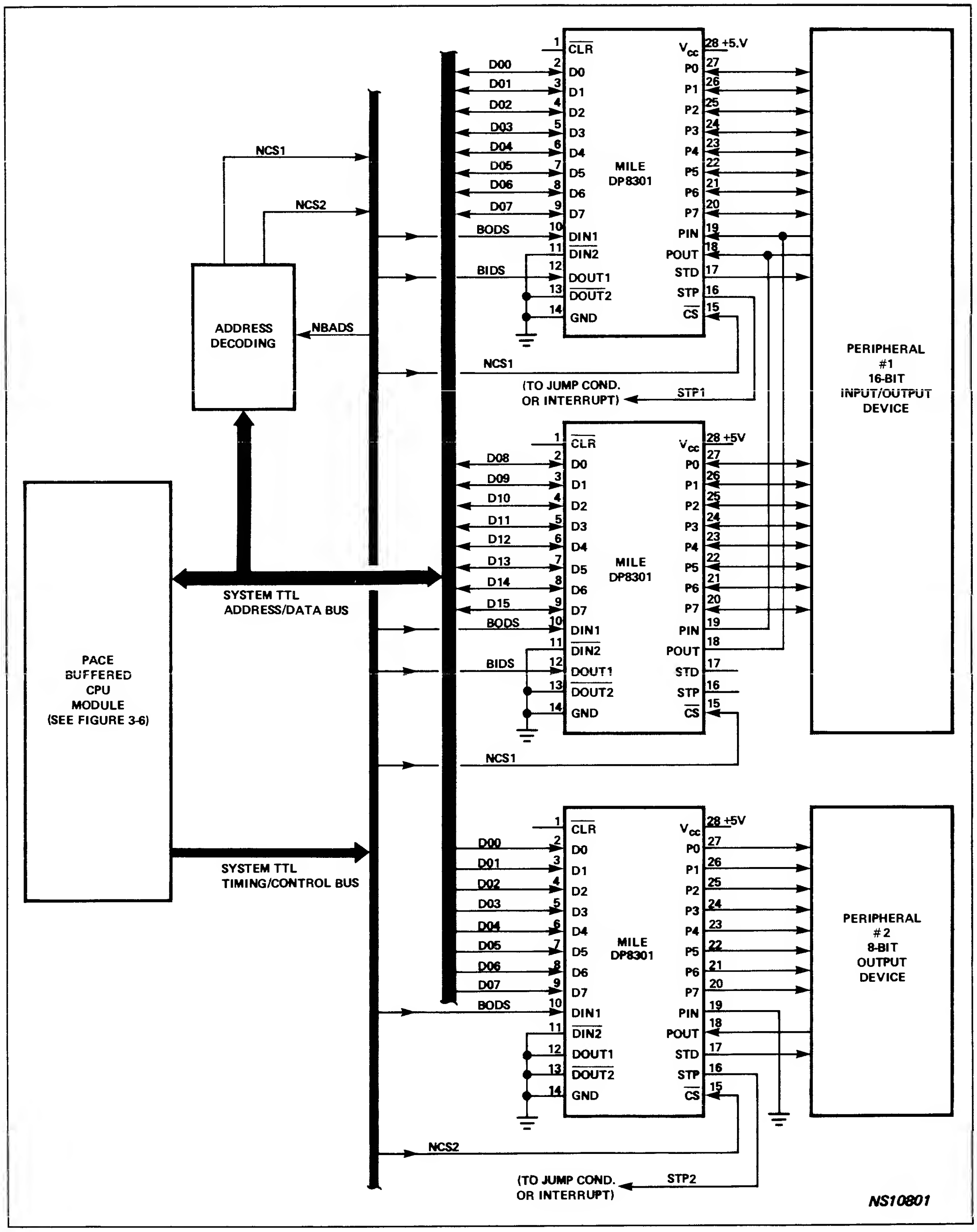
5.3 SPECIAL CONSIDERATIONS WHEN INTERFACING 8-BIT PERIPHERALS

A system using an 8-bit configuration for peripheral or memory addressing usually contains a 16-bit instruction memory (typically ROM), an 8-bit data memory (RAM), and 8-bit peripheral Device Interface Elements (ILE). The instruction memory is 16-bits wide because PACE operation always is controlled by 16-bit instructions regardless of the data length serviced. The hardware design incorporated into the PACE concept permits the microprocessor ALU, registers and Stack to manipulate 16-bit memory addresses and instructions while, at the same time, 8-bit data is manipulated in the 8-bit data mode. Thus, the PACE concept provides greater execution speeds and more powerful instructions for either 8-bit or 16-bit operations than can be achieved with conventional 8-bit microprocessors.

Selection of the 8-bit data-handling capability is accomplished by setting the microprocessor BYTE Status Flag high. When the BYTE Status Flag is set high, PACE execution of Shift and Rotate Instructions is modified and the operation of some of the status flags is changed. In systems servicing both 8-bit and 16-bit data, the user-generated software can incorporate the Set Flag and Pulse Flag Instructions to change the state of the BYTE Status Flag to accommodate the interfaced data length.

5.3.1 Hardware Considerations

Eight-bit peripheral interfacing is described from the hardware viewpoint, along with the corresponding use of PACE control signals, by figure 5-2 and the associated text. The main hardware consideration for 8-bit memory or peripheral interfacing is to ascertain that the eight data lines from the peripheral interface or



PACE BUFFERED CPU MODULE (SEE FIGURE 3-6)
ADDRESS DECODING
NCS1
NCS2
NBADS
SYSTEM TTL ADDRESS/DATA BUS
SYSTEM TTL TIMING/CONTROL BUS
MILE DP8301
CLR
D0
D1
D2
D3
D4
D5
D6
D7
DIN1
DIN2
DOUT1
DOUT2
GND
Vcc
P0
P1
P2
P3
P4
P5
P6
P7
PIN
POUT
STD
STP
CS
+5V
D00
D01
D02
D03
D04
D05
D06
D07
D08
D09
D10
D11
D12
D13
D14
D15
BODS
BIDS
(TO JUMP COND. OR INTERRUPT)
STP1
STP2
PERIPHERAL #1 16-BIT INPUT/OUTPUT DEVICE
PERIPHERAL #2 8-BIT OUTPUT DEVICE
NS10801

Figure 5-2. Typical System Implementation of MILEs

memory are connected to the eight low-order PACE data lines (D00–D07). However, aside from hardware and control signals, some special considerations should be observed for 8-bit memory or peripheral interfacing as regards memory addressing, the PACE status flags, and some of the PACE instructions. These considerations are described in the paragraphs that follow.

5.3.2 Software Considerations

Both the indexed and base-page memory addressing modes require consideration when 8-bit data is processed. Accessing both 16-bit (program) and 8-bit (data) words by using the base-page mode may be desirable. Since two different memories (ROM and RAM) are used, splitting the base page between the two memories may also be desirable. Chapter 4 provides an example of how base-page splitting can be easily accomplished.

For indexed addressing, Accumulators AC2 and AC3 are used as 16-bit memory pointers. If Accumulators AC2 and AC3 are loaded from the 8-bit memory, the high-order 8 bits in the accumulators can be set equal to the sign of the low-order 8 bits by using the Load With Sign Extended Instruction (LSEX). Thus, a 16-bit twos-complement number results.

The Load With Sign Extended Instruction also can be used to set the state of the eight high-order data bits during 8-bit data transfers from peripherals. Alternatively, user-generated software can use Shift Instructions to set the eight high-order data bits to zero. The Shift and Rotate Instruction group (SHL, SHR, ROL, ROR) operates on the low-order 8 bits only and sets the high-order 8 bits to zero when the BYTE Status Flag is set for the 8-bit data-handling mode.

The Immediate Instructions (LI, CAI, AISZ) provide 16-bit, twos-complement data inputs. When working with 8-bit data, the high-order 8 bits usually can be ignored. If required, the high-order 8 bits can be cleared by using a Shift Instruction.

The Branch and Skip Instructions are modified to account for the 8-bit data length. Thus, the REQ0 and NREQ0 conditions are affected only by the low-order 8 bits. The PSIGN and NSIGN Signals indicate the sign of the low-order 8 bits. The Skip Instructions (SKNE, SKG, SKAZ, ISZ, DSZ) test only the low-order 8 bits. Thus, if a Skip Instruction compares 8-bit accumulator data with a 16-bit program memory word, the contents of the high-order 8 bits of both words are ignored. The Add Immediate, Skip if Zero Instruction (AISZ) is the only instruction that tests the entire 16-bit result when 8-bit data handling is selected. Therefore, the AISZ Instruction can be used to increment the index accumulators (AC2, AC3) without skipping every time the low-order 8 bits are zero. Consequently, the sign of 8-bit numbers must be extended by using the Load With Sign Extended Instruction to properly detect zero when using the AISZ Instruction for 8-bit data.

Since the Overflow and Carry Flags are modified by arithmetic instructions, the eight low-order data bits determine the state of the Overflow and Carry Flags when the 8-bit data length is selected. That is, the Carry Flag is set if a carry is generated by the low-order 8 bits and the Overflow Flag is set when an arithmetic overflow occurs in the low-order 8 bits.

The Link Flag is affected by Shift and Rotate Instructions. The Link Flag is set by data shifted out of the low-order 8 bits when the 8-bit data length is selected.

Working with 8-bit data and 16-bit instructions sometimes necessitates performing arithmetic operations by using a 16-bit operand from the program memory and an 8-bit operand from the data memory. If the result is to be treated as 8-bit data, no special considerations are required. However, if the result is to be treated as 16-bit data, the sign of the 8-bit operand first must be extended by using the Load With Sign Extended Instruction. Also, the carry, overflow, and conditional branch signals that are only a function of the low-order 8 bits should not be used. Alternatively, the BYTE Flag temporarily may be set low for 16-bit data handling to accommodate the signals changed by the 8-bit data-handling mode.

The previously mentioned factors make the use of PACE in 8-bit applications convenient while still providing the advantages of a 16-bit instruction set. (Data lengths other than 8 bits or 16 bits also may be used when special external hardware is provided.)

5.4 INTERFACING WITH SLOW PERIPHERALS

For the purposes of this discussion, a 'slow' peripheral is defined as one that cannot capture data from or present data to the system data bus within the constraints of the PACE input/output cycle time. (Refer to 2.4 for a description of PACE input/output operations.) The method of interfacing to slow peripherals depends on the peripheral device characteristics, system demands, and, finally, exactly how slow the peripheral is.

If the peripheral data transfer time exceeds only slightly the timing constraints of PACE, the PACE Extend Signal input can be used to lengthen the input/output cycle. The Extend Signal is described in 2.4.3, and an example of its use with slow memory devices is provided in 4.3. The description of Extend usage with memory devices applies equally to peripheral device usage of this signal.

If the peripheral data transfer is a great deal slower than the PACE input/output cycle time, the use of Extend may not be practical and other approaches should be evaluated.

One alternate method is to use some type of test-and-branch program-controlled input/output. In this method, PACE would output data to the device (or to some intermediate element such as an input/output port). When the device has captured and/or utilized the data and is able to accept additional data, it could then use a jump-condition input to PACE to indicate its readiness. This method requires that PACE (under control of the user's program) test the jump-condition input (using a Branch-On-Condition Instruction) to determine when additional data can be transferred. A similar scheme would be used for data input.

NOTE

The status outputs of the MILE described in 5.2 can be used as inputs to the PACE jump-condition or interrupt-request inputs.

One potential disadvantage of the program-controlled method of input/output just described is that it may require dedication of PACE for an unacceptable amount of time to service the peripheral data transfer. For applications where this is unacceptable, interrupt-driven input/output or direct-memory-access techniques may be more advantageous.

5.5 PERIPHERAL CONTROL

Communications between PACE and peripheral devices typically require, in addition to transferring data words, that control information be sent to the peripheral to establish set-up conditions and operating modes, to initiate and terminate mechanical operations, and so on. While predefined sequences of data words can be used to accomplish these control functions, a variety of other methods can also be used to implement efficient peripheral interfaces.

For example, not all 16 bits of the address word (sent out at the beginning of each input/output operation) are usually required to select a peripheral device. Thus, it may be advantageous to assign part of the address word as a command or control field. Figure 5-3 shows an address word where the 3 least significant bits have been defined as an order code for a peripheral. Typical order codes that might be used are shown in table 5-1.

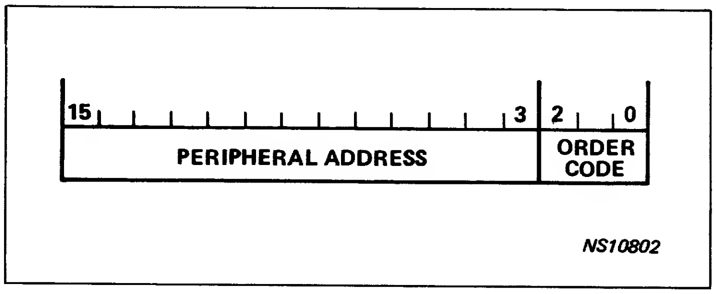


Figure 5-3. Using Address Word for Control

Table 5-1. Possible Order Codes for a Cassette or Tape Drive

ORDER CODE			DEFINITION
BITS			
2	1	0	
0	0	0	Input Character
0	0	1	Output Character
0	1	0	Reset Device
0	1	1	Backspace 1 Record
1	0	0	Advance to Start of TEXT CHAR
1	0	0	Input Device Status
1	1	0	Loop Tape
1	1	1	Test Read Head

Another obvious method of sending control information to peripheral devices is to use the PACE general-purpose control flag outputs (F11–F14). This method may be preferable in some systems since it avoids the need for address decoding and reduces the number of interface lines. The control flag outputs can be set, reset, or pulsed using Set Flag and Pulse Flag Instructions.

NOTE

Refer to 5.7 for a discussion of methods for expanding the number of control flag outputs beyond the four that are directly available from PACE.

5.6 SERIAL INPUT/OUTPUT

Serial interfaces to PACE can be provided by using a single bit of the address/data bus for the interface. Another method, which may be preferable for use in systems containing only a few peripherals, is to use jump condition (JC13, JC14, or JC15) or interrupt inputs (NIR2 through NIR5) for serial input and control flag outputs (F11 through F14) for serial output. This method avoids the need for address decoding and reduces the number of interface lines and is particularly well-suited for asynchronous serial devices such as a teletypewriter, since only one transmit and one receive line are involved. Use of interrupts is described in 5.8; use of jump condition inputs and control flag outputs are described briefly in the paragraph that follows.

A simple teletypewriter serial interface can be implemented using one jump-condition input and one control-flag output. Figure 5-4 shows the circuit required to use the Buffered Flag 11 (BF11) output from PACE (via a BTE) to output serial data to a 20-milliampere current-loop interface and the circuitry that could be used to accept input data from a teletypewriter via the Jump Condition 13 (JC13) input to PACE. The remainder of the serial input/output is handled by a user-supplied program that would furnish the required timing and testing operations to assure the transfer of data at the appropriate bit rate.

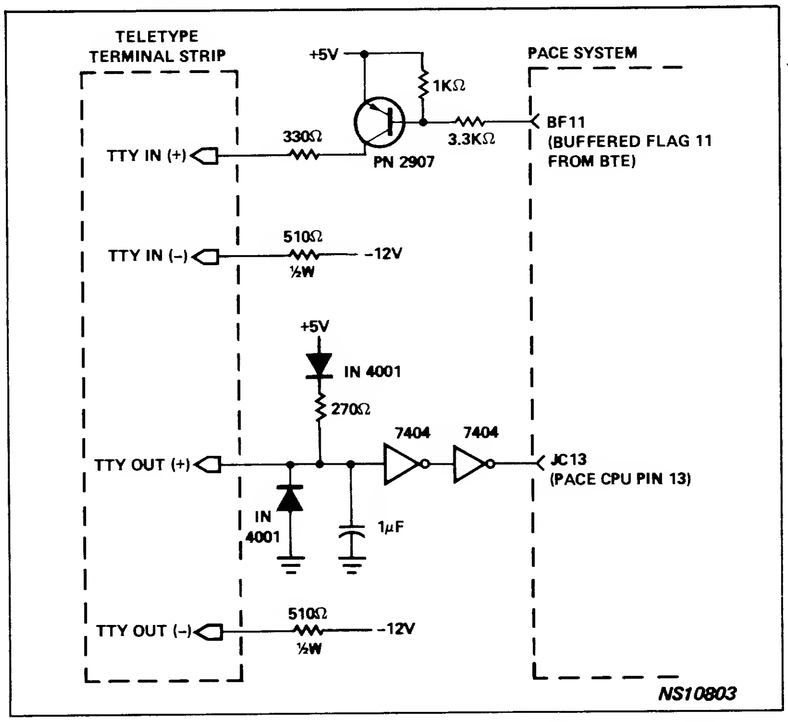


Figure 5-4. PACE/Teletype Interface Circuit

5.7 EXPANDING JUMP CONDITIONS AND FLAGS

Four user flags are provided by the PACE Central Processing Unit. In some cases, additional flags may be required for control purposes. The additional flags can be conveniently obtained by using a DM9334 8-bit addressable latch. An unused address bit or combination of bits may be utilized to enable the latch; 3 bits then can address one of eight flags and another bit can specify set or reset as illustrated in figure 5-5. A Store Instruction (ST) may be used to output the address (data output is ignored).

In a similar manner, a multiplexer and latch can be utilized to expand the user jump conditions. The latch is loaded from the address bus if enabled by an unused address code and seleects a multiplexer input to one of the user jump conditions.

5.8 INTERRUPT DRIVEN INPUT/OUTPUT

The PACE interrupt system can save considerable hardware and software in applications containing several interrupts by virtue of the on-chip priority logic and vectored branch to the interrupt routine. No external logic circuit is required to resolve priority and jam an address vector onto the data bus as with many other microprocessors. Also, interrupt program storage and execution time are saved, since sequential polling of the interrupt status of all devices is not required.

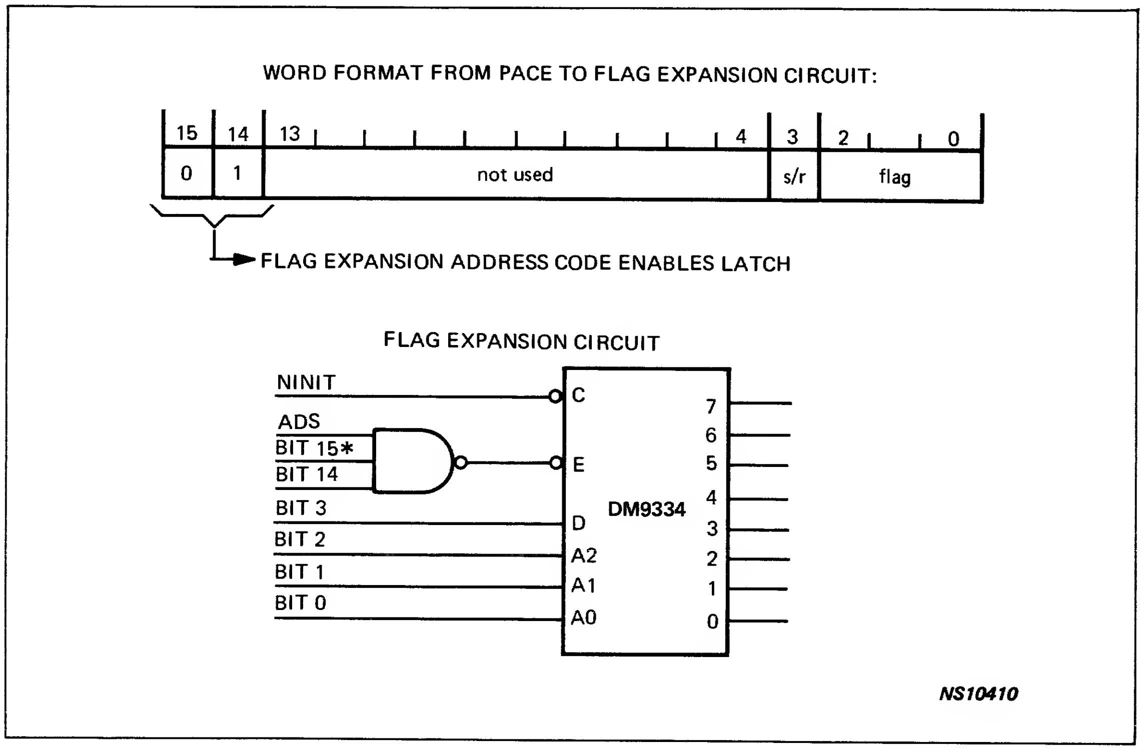


Figure 5-5. One Possible Circuit and Word Format for Obtaining Additional Flags

Interrupts are essential in applications where alarm conditions or transient conditions (such as automobile, process, or machine tool control) must be serviced immediately. Interrupts are useful in many other systems to eliminate the program overhead required to scan asynchronous system inputs (such as a controller for multiple terminals or an intersection traffic light controller).

A detailed description of how interrupts are handled internally by PACE is provided in 2.7. The paragraphs that follow describe additional considerations that apply to application systems using the PACE interrupt structure.

Three types of external interrupts are likely to occur in PACE applications: short-duration (pulse) interrupts, long-duration resettable interrupts, and nonresettable interrupts. The short-duration interrupt exists for less than the interrupt response time and may be caused by a strobe pulse from a peripheral device or the occurrence of a high-speed transient condition. A short-duration interrupt must be latched to be recognized. Interrupts longer than the clock period are latched by the PACE Interrupt Request Latches. The Interrupt Service Routine must reset the Interrupt Request Latch by turning off the Interrupt Enable for the level being serviced. If the Interrupt Enable is left off, interrupt request pulses cannot set the Interrupt Request Latch.

Long-duration resettable interrupts last longer than the interrupt response time and may be reset by the Interrupt Service Routine. An example is a buffer-full interrupt by a peripheral device. The Interrupt Service Routine empties the buffer, removing the interrupt. A long-duration interrupt is ignored when the Interrupt Enable Signal is low but still generates an interrupt when the Interrupt Enable Signal is set high. In servicing long-duration interrupts, the Interrupt Request Latch must be cleared after the interrupt is reset by the Interrupt Service Routine.

Long-duration nonresettable interrupts last longer than the interrupt response time and are not reset by the Interrupt Service Routine. An example of a long-duration nonresettable interrupt is a photoelectric cell that detects the presence of an item on a conveyor. The signal produced by the photoelectric cell (or some other sensor) may last for a significant portion of a second. Setting the Interrupt Request Latch on the edge of the interrupt is desirable and may be accomplished using a simple RC circuit or a single-shot to generate a pulse on the edge of the interrupt.

The interrupt response time for PACE is equal to the time to finish the current instruction at the time of the interrupt, plus the time to access the first instruction of the Interrupt Service Routine.

NOTE

Refer to the PACE Programming and Assembly Language Manual for examples of interrupt service routines.

5.9 EXPANDING THE INTERRUPT SYSTEM

In some applications, expansion of the user interrupts by providing several interrupts on a single input may be desirable. Several interrupts can be provided on a single input by using open-collector gates for a wired-OR input and polling all the devices on a given level to discover the origin of the interupt. However, in some applications, the polling technique may take excessive time. In such cases, use of the DM9318 Priority Encoder is recommended to encode the number of the highest-priority interrupting device. A single instruction in the Interrupt Service Routine then can be used to read the number of the interrupting device over the data bus. The use of the DM9318 Priority Encoder is shown in figure 5-6. The use of a DM8131 Comparator with latched output to detect the appropriate peripheral address also is illustrated in figure 5-6.

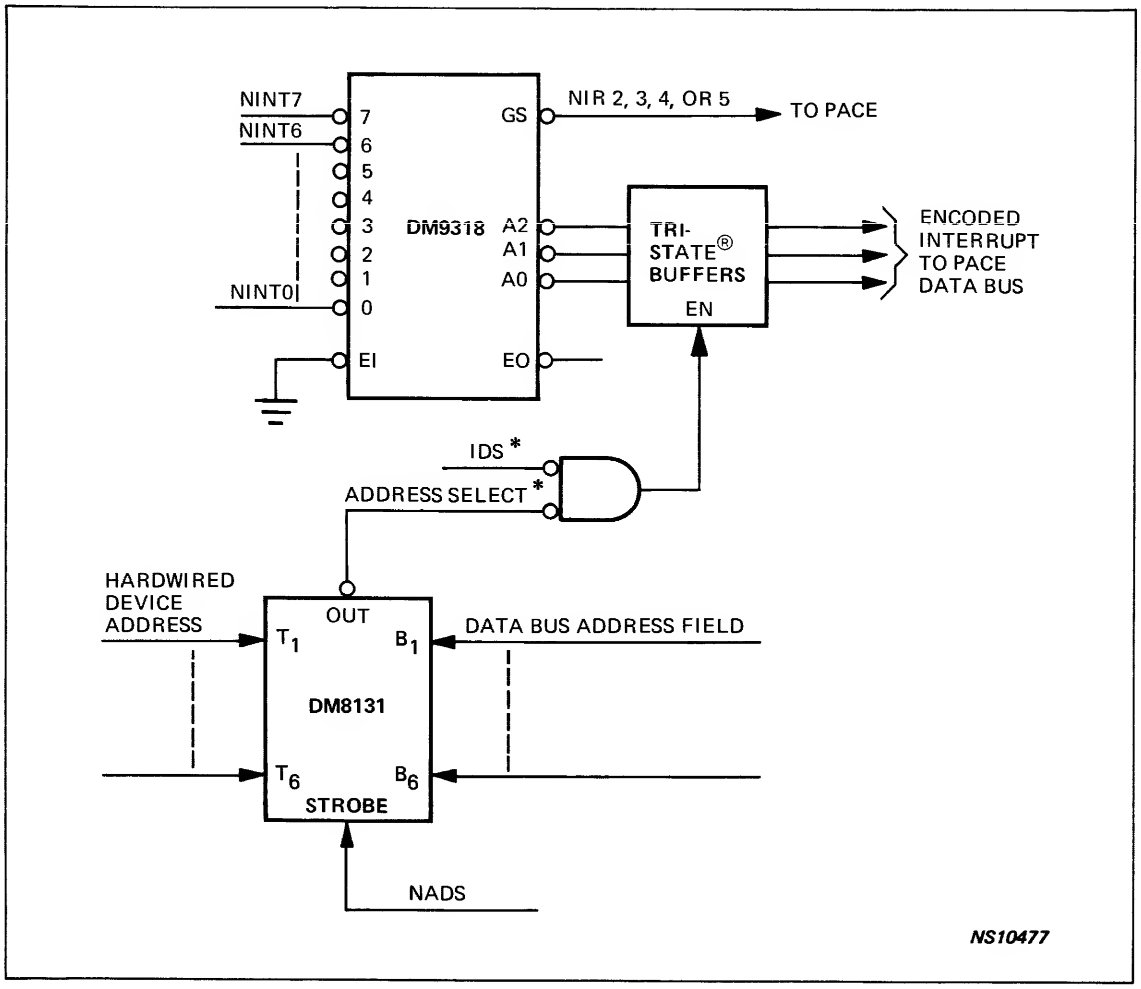


Figure 5-6. Use of DM9318 Priority Encoder and DM8131 Comparator for Interrupt Expansion and Detection

Appendix A

PACE INSTRUCTION RELATED SUMMARIES

A.1 INTRODUCTION

Table A-1 defines the notation and symbols used for the symbolic representation of each instructon contained in table A-2, the Detailed Instruction Summary. The notations in table A-1 are presented in alphabetical order and, then, the symbols are listed. Upper-case mnemonics refer to fields in the instruction word. Lower-case mnemonics refer to the numerical value of the corresponding fields. In cases where both upper-case and lower-case mnemonics are composed of the same letters, only the lower-case mnemonic is given. The use of lower-case notation designates variables.

The formulas in table A-2 for computing the execution times of instructions are presented in terms of machine (microinstruction) cycles (M) and input/output data-transfer Cycle Extends (E_R for read and E_W for write). Each machine cycle (M) consists of four clock periods (t_p). The Cycle Extend (E_R, E_W) factors need be included only if operating with slow memory or peripheral devices that require the use of the PACE EXTEND Signal (refer to sections 2.4.3, 4.3, and 5.4). The following example shows the method to be used for computing the execution times of instructions.

Example:

The formula (listed in table A-2) for the execution time of a RADD Instruction is $4M + E_R$. If the clock period is 750 nanoseconds and no Read Cycle Extend is required, then:

$M = 4\ (0.75\ \mu\text{sec}) = 3\ \mu\text{sec}$
$E_R = 0$

therefore: $4M + E_R = 4\ (3\ \mu\text{sec}) + 0 = 12\ \mu\text{sec}$. Thus, under the hypothetical clock cycle and Read Cycle Extend times used, the RADD Instruction requires 12 microseconds for execution.

Following the detailed instruction summary (table A-2, an Op Code Index of Instructions is provided in table A-3. In this table, the instructions are listed in alphanumeric sequence (by hexadecimal) according to the value of each instruction's opcode. This index is useful when hand-coding and debugging programs.

Table A-1. Notations/Symbols Used in Instruction Descriptions

Notation/ Symbol	Meaning
ACr	Denotes specific working accumulator (AC0, AC1, AC2, or AC3), where *r* is number of accumulator referenced in instruction.
B	Denotes instruction execution affected by state of Byte Flag.
cc	Denotes 4-bit condition code value for conditional branch instructions.
CRY	Indicates Carry Flag is set if carry exists due to instruction (either addition or subtraction) or reset if no carry exists.
disp	Stands for displacement value and represents operand in non-memory-reference instruction or address field in memory-reference instruction. Disp is 8-bit, signed twos-complement number except when base page is referenced; in latter case, disp is unsigned if BPS=0.
dr	Denotes number of destination working accumulator specified in instruction-word field. Working accumulator is AC0, AC1, AC2, or AC3.
EA	Denotes effective address specified by instructions directly, indirectly, or by indexing. Effective address contents are used during execution of instruction.
fc	Denotes number of referenced flag.
FR	Denotes Status and Control Flags Register.
IEN	Denotes Interrupt Enable Flag.
ℓ.	Denotes inclusion of 1-bit Link Flag (LINK) in shift operations.
n	Unsigned number indicates number of bit positions to be shifted in Shift and Rotate Instructions.
OVF	Indicates Overflow Flag is set if overflow exists due to instruction (either addition or subtraction) or is reset if no overflow exists. Overflow occurs if signs of operands are alike and sign of result is different from operands.
PC	Denotes Program Counter. During address formation, PC is incremented by 1 to contain address 1 greater than that of instruction being executed.
r	Denotes number of working accumulator specified in instruction-word field. Working accumulator is AC0, AC1, AC2, or AC3.
STK	Denotes top word of 10-word Last-In/First-Out-Stack.
sr	Denotes number of source working accumulator specified in instruction-word field. Working accumulator is AC0, AC1, AC2, or AC3.

Table A-1. Notations/Symbols Used in Instruction Descriptions (Concluded)

Notation/ Symbol	Meaning
xr	When not zero, xr value designates number of accumulator to be used in indexed and relative-memory addressing modes. When zero, base-page addressing is indicated.
( )	Denotes contents of item within parentheses. (ACr) is read as *contents of ACr.* (EA) is read as *contents of EA.*
[ ]	Denotes *results of.*
$\sim$	Indicates logical complement (ones' complement) of value on right-hand side of $\sim$.
$\rightarrow$	Means *replace.*
$\leftarrow$	Means *is replaced by.*
@	Appearing in operand field of instruction, denotes indirect addressing.
+10	Modulo 10 addition.
$\wedge$	Denotes AND operation.
$\vee$	Denotes OR operation.
$\veebar$	Denotes EXCLUSIVE OR operation.

Table A-2. PACE Instruction Summary

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
BRANCH INSTRUCTIONS			
Branch-On Condition BOC 15–12: 0 1 0 0 | 11–08: cc | 07–00: disp	If cc true, (PC)←(PC) + disp, B 16 possible condition codes (cc) exist. Condition codes are listed in table B-3. If condition for branching designated by cc is true, value of disp (sign extended from bit 7 through bit 15) is added to PC and sum is stored in PC.	BOC cc, disp	5M + E_R + 1M if branch
Jump JMP 15–10: 0 0 0 1 1 0 | 09–08: xr | 07–00: disp	(PC)←EA Effective address EA replaces PC contents. Next instruction is fetched from location designated by new contents of PC.	JMP disp (xr)	4M + E_R
Jump Indirect JMP@ 15–10: 1 0 0 1 1 0 | 09–08: xr | 07–00: disp	(PC)←(EA) Contents of effective address replace PC contents. Next instruction is fetched from location designated by new contents of PC.	JMP @disp (xr)	4M + 2E_R
Jump to Subroutine JSR 15–10: 0 0 0 1 0 1 | 09–08: xr | 07–00: disp	(STK)←(PC), (PC)←EA Contents of PC are stored on top of Stack. Effective address replaces PC contents. Next instruction is fetched from location designated by new contents of PC.	JSR disp (xr)	5M + E_R
Jump to Subroutine Indirect JSR@ 15–10: 1 0 0 1 0 1 | 09–08: xr | 07–00: disp	(STK)←(PC), (PC)←(EA) Contents of PC are stored on top of Stack. Contents of effective address replace PC contents. Next instruction is fetched from location designated by new contents of PC.	JSR @disp (xr)	5M + 2E_R
Return from Subroutine RTS 15–08: 1 0 0 0 0 0 0 0 | 07–00: disp	(PC)←(STK) + disp Contents of PC are replaced by sum of disp added to contents pulled from top of Stack. Program control is transferred to location specified by new contents of PC.	RTS disp	5M + E_R

Table A-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
BRANCH INSTRUCTIONS (Continued)			
Return from Interrupt RTI — 15 08 07 00: 0 1 1 1 1 1 0 0 disp	(PC)←(STK) + disp, IEN = 1. Interrupt Enable Flag (IEN) is set. PC contents are replaced by sum of disp and word pulled from top of Stack. Program control is transferred to location specified by new contents of PC.	RTI disp	6M + E_R
SKIP INSTRUCTIONS			
Skip if Not Equal SKNE — 15 12 11 10 09 08 07 00: 1 1 1 1 r xr disp	If (ACr) ≠ (EA), (PC)←(PC) + 1, B. ACr contents and contents of effective memory location EA are compared. If contents of ACr and EA are not equal, next instruction in sequence is skipped. Contents of ACr and EA are unaltered. If 8-bit data length is selected, only lower 8 bits are compared.	SKNE r, disp (xr)	5M + $2E_R$ + 1M if skip
Skip if Greater SKG — 15 10 09 08 07 00: 1 0 0 1 1 1 xr disp	If (AC0) > (EA), (PC)←(PC) + 1, B. AC0 contents and contents of effective memory location EA are compared as signed numbers. If contents of AC0 are greater (more positive) than contents of EA, next instruction in sequence is skipped. Contents of AC0 and EA are unaltered. If 8-bit data length is selected, only lower 8 bits are compared.	SKG 0, disp (xr)	7M + $2E_R$ + 1M if skip
Skip if AND is Zero SKAZ — 15 10 09 08 07 00: 1 0 1 1 1 0 xr disp	If ((AC0) ∧ (EA)) = 0, (PC)←(PC) + 1, B. AC0 contents and contents of effective memory location EA are ANDed. If result equals zero, next instruction in sequence is skipped. Contents of AC0 and EA are unaltered. If 8-bit data length is selected, only lower 8 bits are tested.	SKAZ 0, disp (xr)	5M + $2E_R$ + 1M if skip

Table A-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
SKIP INSTRUCTIONS (Continued)			
Increment and Skip if Zero ISZ 15 10 | 09 08 | 07 00 1 0 0 0 1 1 | xr | disp	(EA)←(EA) + 1; if (EA) = 0, (PC)←(PC) + 1, B EA contents are incremented by 1. If new contents of EA equal zero, next instruction in sequence is skipped. If 8-bit data length is selected, only lower 8 bits are tested.	ISZ disp (xr)	$7M + 2E_R + E_W$ + 1M if skip
Decrement and Skip if Zero DSZ 15 10 | 09 08 | 07 00 1 0 1 0 1 1 | xr | disp	(EA)←(EA) - 1; if (EA) = 0, (PC)←(PC) + 1, B EA contents are decremented by 1. If new contents of EA equal zero, next instruction in sequence is skipped. If 8-bit data length is selected, only lower 8 bits are tested.	DSZ disp (xr)	$7M + 2E_R + E_W$ + 1M if skip
Add Immediate, Skip if Zero AISZ 15 10 | 09 08 | 07 00 0 1 1 1 1 0 | r | disp	(ACr)←(ACr) + disp; if (ACr) = 0, (PC)←(PC) + 1 ACr contents are replaced by sum of contents of ACr and disp (sign bit 7 extended through bit 15). Initial contents of ACr are lost. If new contents of ACr equal zero, contents of PC are incremented by 1, thus skipping next instruction. AISZ Instruction always tests full 16-bit result independent of data length selected.	AISZ r, disp	$5M + E_R$ + 1M if skip
MEMORY DATA-TRANSFER INSTRUCTIONS			
Load LD 15 12 | 11 10 | 09 08 | 07 00 1 1 0 0 | r | xr | disp	(ACr)←(EA) ACr contents are replaced by EA contents. Initial contents of ACr are lost; contents of EA are unaltered.	LD r, disp (xr)	$4M + 2E_R$

Table A-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
MEMORY DATA-TRANSFER INSTRUCTIONS (Continued)			
Load Indirect LD@ 15 10 | 09 08 | 07 00 1 0 1 0 0 0 | xr | disp	$(AC0) \leftarrow ((EA))$ AC0 contents are replaced indirectly by EA contents. Initial contents of AC0 are lost; contents of EA and location designating EA are unaltered.	LD 0, @disp (xr)	$5M + 3E_R$
Store ST 15 11 10 | 09 08 | 07 00 1 1 0 1 r | xr | disp	$(EA) \leftarrow (ACr)$ EA contents are replaced by contents of ACr. Initial contents of EA are lost; contents of ACr are unaltered.	ST r, disp (xr)	$4M + E_R + E_W$
Store Indirect ST@ 15 10 | 09 08 | 07 00 1 0 1 1 0 0 | xr | disp	$((EA)) \leftarrow (AC0)$ EA contents are replaced indirectly by AC0 contents. Initial contents of EA are lost; contents of AC0 and location designating EA are unaltered.	ST 0, @disp (xr)	$4M + 2E_R + E_W$
Load with Sign Extended LSEX 15 10 | 09 08 | 07 00 1 0 1 1 1 1 | xr | disp	$(AC0) \leftarrow (EA)$ sign extended AC0 contents are replaced by EA contents with bit 7 extended through bits 8-15. Initial contents of AC0 are lost; contents of EA are unaltered. LSEX permits 8-bit data loading from memory or peripheral to be operated on as 16-bit data.	LSEX 0, disp (xr)	$4M + 2E_R$
MEMORY DATA-OPERATE INSTRUCTIONS			
AND AND 15 10 | 09 08 | 07 00 1 0 1 0 1 0 | xr | disp	$(AC0) \leftarrow (AC0) \wedge (EA)$ AC0 contents and EA contents are ANDed. Result is stored in AC0. Initial contents of AC0 are lost; contents of EA are unaltered.	AND 0, disp (xr)	$4M + 2E_R$

Table A-2. PACE Instruction Summary (Continued)

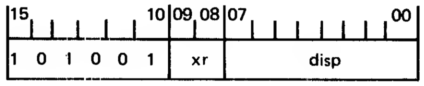
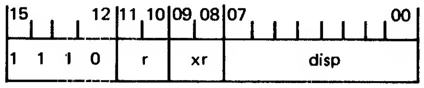
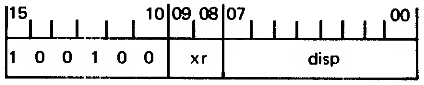
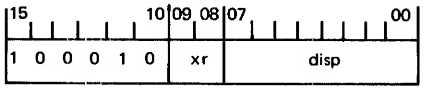
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
MEMORY DATA-OPERATE INSTRUCTIONS (Continued)			
OR OR 	$(AC0) \leftarrow (AC0) \vee (EA)$ AC0 contents and EA contents are ORed inclusively. Result in stored in AC0. Initial contents of AC0 are lost; contents of EA are unaltered.	OR 0, disp (xr)	$4M + 2E_R$
Add ADD 	$(ACr) \leftarrow (ACr) + (EA)$, OV, CRY, B ACr contents are added algebraically to EA contents. Sum is stored in ACr, and contents of EA are unaltered. Initial contents of ACr are lost. Overflow or Carry Flag is set if overflow or carry occurs, respectively; otherwise Overflow and Carry Flags are cleared.	ADD r, disp (xr)	$4M + 2E_R$
Subtract with Borrow SUBB 	$(AC0) \leftarrow (AC0) + \sim(EA) + (CRY)$, OVF, CRY, B AC0 contents are added to complement of EA and carry. Result is stored in AC0 and contents of EA are unaltered. Initial contents of AC0 are lost. Carry and Overflow Flags are set according to result of operation.	SUBB 0, disp (xr)	$4M + 2E_R$
Decimal Add DECA 	$(AC0) \leftarrow (AC0) +_{10} (EA) +_{10} (CRY)$, OVF, CRY, B AC0 contents are treated as 4-digit number and added modulo 10 (for each digit) to contents of EA (treated as 4-digit number) and carry. Initial contents of AC0 are lost; contents of EA are unaltered. Carry Flag is set based on decimal carry output. Overflow Flag is set to arbitrary state.	DECA 0, disp (xr)	$7M + 2E_R$

Table A-2. PACE Instruction Summary (Continued)

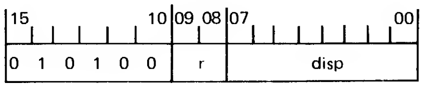
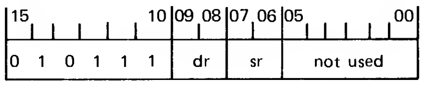
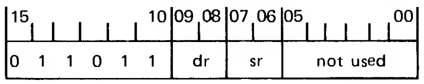
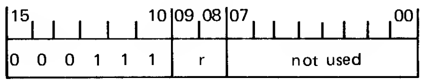
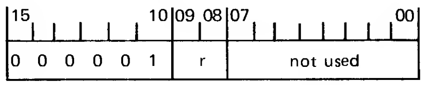
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
REGISTER DATA-TRANSFER INSTRUCTIONS			
Load Immediate LI 	(ACr)←disp ACr contents are replaced by disp with sign bit 7 extended through bit 15. Initial contents of ACr are lost.	LI r, disp	$4M + E_R$
Register Copy RCPY 	(ACdr)←(ACsr) Destination Register ACdr contents are replaced by contents of Source Register ACsr. Initial contents of ACdr are lost and initial contents of ACsr are unaltered.	RCPY sr, dr	$4M + E_R$
Register Exchange RXCH 	(ACdr)←(ACsr), (ACsr)←(ACdr) ACsr contents and ACdr contents are exchanged.	RXCH sr, dr	$6M + E_R$
Exchange Register and Stack XCHRS 	(STK)←(ACr), (ACr)←(STK) Contents of top of Stack and accumulator designated by ACr are exchanged.	XCHRS r	$6M + E_R$
Copy Flags into Register CFR 	(ACr)←(FR) ACr contents are replaced by contents of FR. Initial contents of ACr are lost; contents of FR are unaltered.	CFR r	$4M + E_R$

Table A-2. PACE Instruction Summary (Continued)

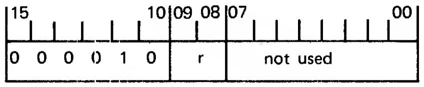
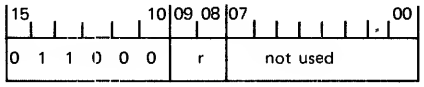
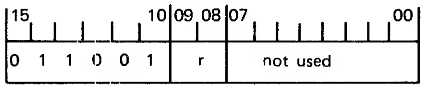
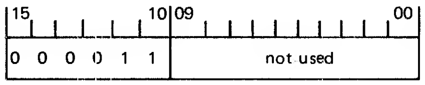
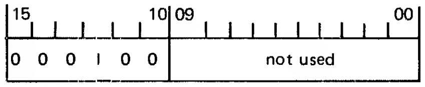
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
REGISTER DATA-TRANSFER INSTRUCTIONS (Continued)			
Copy Register into Flags CRF 	(FR)←(ACr) FR contents are replaced by ACr contents. Initial contents of FR are lost; contents of ACr are unaltered.	CRF r	4M + E_R
Push Register onto Stack PUSH 	(STK)←(ACr) Stack is pushed by contents of accumulator designated by ACr. Thus, top of Stack holds ACr contents and Stack Pointer is incremented by 1. Initial contents of ACr are unaltered.	PUSH r	4M + E_R
Pull Stack into Register PULL 	(ACr)←(STK) Stack is pulled. Contents from top of Stack replace ACr contents. Initial contents of ACr are lost. Contents of Stack Pointer are decremented by 1.	PULL r	4M + E_R
Push Flags onto Stack PUSHF 	(STK)←(FR) FR contents are pushed onto Stack. Contents of FR are unchanged.	PUSHF	4M + E_R
Pull Stack into Flags PULLF 	(FR)←(STK) FR contents are replaced by contents pulled from top of Stack. Initial contents of FR are lost.	PULLF	4M + E_R

Table A-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
REGISTER DATA-OPERATE INSTRUCTIONS			
Register Add RADD 15 10 | 09 08 | 07 06 | 05 00 0 1 1 0 1 0 | dr | sr | not used	(ACdr)←(ACdr) + (ACsr), OVF, CRY, B ACdr contents are replaced by sum of contents of ACdr and ACsr. Initial contents of ACdr are lost and contents of ACsr are unaltered. Overflow and Carry Flags are modified according to result.	RADD sr, dr	$4M + E_R$
Register Add with Carry RADC 15 10 | 09 08 | 07 06 | 05 00 0 1 1 1 0 1 | dr | sr | not used	(ACdr)←(ACdr) + (ACsr) + (CRY), OVF, CRY, B ACdr contents are replaced by sum of ACdr and ACsr contents and carry. Initial ACdr contents are lost and ACsr contents are unaltered. Overflow and Carry Flags are modified according to result.	RADC sr, dr	$4M + E_R$
Register AND RAND 15 10 | 09 08 | 07 06 | 05 00 0 1 0 1 0 1 | dr | sr | not used	(ACdr)←(ACdr) $\wedge$ (ACsr) ACdr contents are replaced by result of ANDing ACdr and ACsr contents. Initial contents of ACdr are lost and initial contents of ACsr are unaltered.	RAND sr, dr	$4M + E_R$
Register EXCLUSIVE OR RXOR 15 10 | 09 08 | 07 06 | 05 00 0 1 0 1 1 0 | dr | sr | not used	(ACdr)←(ACdr) $\veebar$ (ACsr) ACdr contents are replaced by result of EXCLUSIVEly ORing ACdr contents and ACsr contents. Initial contents of ACdr are lost and initial contents of ACsr are unaltered.	RXOR sr, dr	$4M + E_R$
Complement and Add Immediate CAI 15 10 | 09 08 | 07 00 0 1 1 1 0 0 | r | not used	(ACr)←~(ACr) + disp ACr contents are replaced by sum of complement of ACr and disp (sign bit 7 extended through bit 15). Initial contents of ACr are lost. Values of 0 and 1 in disp field produce ones and twos, complement, respectively, of (ACr).	CAI r, disp	$5M + E_R$

Table A-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
SHIFT AND ROTATE INSTRUCTIONS			
Shift Left SHL 15 10 | 09 08 | 07 01 | 00 0 0 1 0 1 0 | r | n | ℓ	(ACr)←(ACr) shifted left n places, w/wo link, B ACr contents are shifted left n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Data shifted out of most significant bit for specified data length are lost if ℓ = 0 and are loaded into LINK if ℓ = 1.	SHL r, n, ℓ	$(5 + 3n)$ M + E_R, n = 1-127; 6M + E_R, n = 0
Shift Right SHR 15 10 | 09 08 | 07 01 | 00 0 0 1 0 1 1 | r | n | ℓ	(ACr)←(ACr) shifted right n places, w/wo link, B ACr contents are shifted right n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Zeros are shifted into most significant bit for specified data length if ℓ = 0. Contents of LINK are shifted in if ℓ = 1, and contents of LINK are unchanged. Data shifted out of least significant bit are lost.	SHR r, n, ℓ	$(5 + 3n)$ M + E_R, n = 1-127; 6M + E_R, n = 0
Rotate Left ROL 15 10 | 09 08 | 07 01 | 00 0 0 1 0 0 0 | r | n | ℓ	(ACr)←(ACr) rotated left n places, w/wo link, B ACr contents are rotated left n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Data shifted out of most significant bit position for specified data length are shifted into least significant bit if ℓ = 0, and into LINK if ℓ = 1, in which case least significant bit is loaded from LINK.	ROL r, n, ℓ	$(5 + 3n)$ M + E_R, n = 1-127; 6M + E_R, n = 0
Rotate Right ROR 15 10 | 09 08 | 07 01 | 00 0 0 1 0 0 1 | r | n | ℓ	(ACr)←(ACr) rotated right n places, w/wo link, B ACr contents are rotated right n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Data shifted out of least significant bit are shifted into most significant bit for specified data length if ℓ = 0, and into LINK if ℓ = 1, in which case most significant bit is loaded from LINK.	ROR r, n, ℓ	$(5 + 3n)$ M + E_R, n = 1-127; 6M + E_R, n = 0

Table A-2. PACE Instruction Summary (Concluded)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
MISCELLANEOUS INSTRUCTIONS			
Halt HALT 15 10 | 09 00 0 0 0 0 0 0 | not used	Halt Microprocessor halts and remains halted until CONTIN Input to Jump Condition Multiplexer makes transition from logic '1' to logic '0'.	HALT	------
Set Flag SFLG 15 12 | 11 08 | 07 | 06 00 0 0 1 1 | fc | 1 | not used	$(FR)_{fc} \leftarrow 1$ Flag, or bit of FR, specified by flag code fc is set true. All other bits of FR are unaltered.	SFLG fc	$5M + E_R$
Pulse Flag PFLG 15 12 | 11 08 | 07 | 06 00 0 0 1 1 | fc | 0 | not used	$(FR)_{fc} \leftarrow 1$, $(FR)_{fc} \leftarrow 0$ Flag (bit fc of FR) is first set true and then set false (after four clock periods), causing pulsing or resetting of flag, depending on initial state of flag. All other bits of FR are unaffected.	PFLG fc	$6M + E_R$

Table A-3. Op Code Index of Instructions

ALPHANUMERIC SEQUENCE BY HEXADECIMAL

Read down then right.

Mnemonic Assembler Code		AC0	AC1	AC2	AC3	BASE PAGE (XX)	PC REL (XX+PC)	AC2 REL (XX+AC2)	AC3 REL (XX+AC3)									
HALT	0000																	Halt
CFR r		0400	0500	0600	0700													Copy flags to register
CRF r		0800	0900	0A00	0800													Copy register to flags
PUSHF	0C00																	Push flags onto stack
PULLF	1000																	Pull stack into flags
JSR disp(xr)						14XX	15XX	16XX	17XX									Jump to subroutine; XX = ±127; push PC onto stack
JMP disp(xr)						18XX	19XX	1AXX	1BXX									Jump; XX = ±127
XCHRS r		1C00	1D00	1E00	1F00													Exchange register and stack
RDL r,n,l		20XX	21XX	22XX	23XX													Rotate register left
RDR r,n,l		24XX	25XX	26XX	27XX													Rotate register right } Bit 1 = 1 include link bit
SHL r,n,l		28XX	29XX	2AXX	2BXX													Shift left } Bit 2 = 2 shift count
SHR r,n,l		2CXX	2DXX	2EXX	2FXX													Shift right } Bits 2-7 = N = shift count
fc		NDT USED	IE1	IE2	IE3	IE4	IE5	DVF	CRY	LINK	IEN	BYTE	F11	F12	F13	F14	NDT USED	
PFLG fc		3000	3100	3200	3300	3400	3500	3600	3700	3800	3900	3A00	3B00	3C00	3D00	3E00	3F00	Pulse or reset flag
SFLG fc		3080	3180	3280	3380	3480	3580	3680	3780	3880	3980	3A80	3880	3C80	3D80	3E80	3F80	Set flag
cc		STACK Full	AC0 = 0	AC0 Bit15=0	AC0 Bit0=1	AC0 Bit1=1	AC0 ≠ 0	AC0 Bit2=1	CONT	LINK	IEN	CRY	AC0 Bit 15=1	DVF	JC13	JC14	JC15	
BDC cc,disp		40XX	41XX	42XX	43XX	44XX	45XX	46XX	47XX	48XX	49XX	4AXX	48XX	4CXX	4DXX	4EXX	4FXX	Branch on condition (PC relative) XX = ±127
		AC0	AC1	AC2	AC3													
LI r, **data**		50XX	51XX	52XX	53XX													Load immediate; load register with XX; XX = data Bit 7 of XX extends to Bits 8-15 of register
sr		AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	
dr		AC0	AC0	AC0	AC0	AC1	AC1	AC1	AC1	AC2	AC2	AC2	AC2	AC3	AC3	AC3	AC3	
RAND sr,dr		5400	5440	5480	54C0	5500	5540	5580	55C0	5600	5640	5680	56C0	5700	5740	5780	57C0	"AND" register to register; result to register (dr)
RXOR sr,dr		5800	5840	5880	58C0	5900	5940	5980	59C0	5A00	5A40	5A80	5AC0	5800	5840	5880	58C0	Exclusive "OR" register to register; result to register (dr)
NOP	**5C00**																	
RCPY sr,dr		5C00	5C40	5C80	5CC0	5D00	5D40	5D80	5DC0	5E00	5E40	5E80	5EC0	5F00	5F40	5F80	5FC0	Copy register to register
		AC0	AC1	AC2	AC3													
PUSH r		6000	6100	6200	6300													Push register onto stack
PULL r		6400	6500	6600	6700													Pull stack into stack
sr		AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	
dr		AC0	AC0	AC0	AC0	AC1	AC1	AC1	AC1	AC2	AC2	AC2	AC2	AC3	AC3	AC3	AC3	
RADD sr,dr		6800	6840	6880	68C0	6900	6940	6980	69C0	6A00	6A40	6A80	6AC0	6800	6840	6880	6BC0	Add register to register; result to register (dr), overflow, and carry
RXCH sr,dr		6C00	6C40	6C80	6CC0	6D00	6D40	6D80	6DC0	6E00	6E40	6E80	6EC0	6F00	6F40	6F80	6FC0	Exchange register
		AC0	AC1	AC2	AC3													
CAI r, **data**		70XX	71XX	72XX	73XX													Complement register and add XX; result to register Bit 7 of XX is extended to Bits 8-15
sr		AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	AC0	AC1	AC2	AC3	
dr		AC0	AC0	AC0	AC0	AC1	AC1	AC1	AC1	AC2	AC2	AC2	AC2	AC3	AC3	AC3	AC3	
RADC sr,dr		7400	7440	7480	74C0	7500	7540	7580	75C0	7600	7640	7680	76C0	7700	7740	7780	77C0	Add register to register plus carry; result to register (dr); overflow and carry

Table A-3. Op Code Index of Instructions (Concluded)

ALPHANUMERIC SEQUENCE BY HEXADECIMAL

Read down then right.

Mnemonic Assembler Code			AC0	AC1	AC2	AC3	BASE PAGE XX	PC REL (XX+PC)	AC2 REL (XX+AC2)	AC3 REL (XX+AC3)	
AISZ	r, data		78XX	79XX	7AXX	7BXX					Add XX to register; skip next instruction if result = zero; XX = ±127
RTI	disp	7CXX									Return from interrupt; add XX to top of stack and place result in PC; XX = ±127; set IEN flag
RTS	disp	80XX									Return from subroutine; add XX to top of stack and place result in PC; XX = ±127
DECA	0, disp(xr)						88XX	89XX	8AXX	8BXX	Decimal add register AC0 to contents of effective address; result to AC0, overflow and carry; address = (XX + register shown); XX = ±127
ISZ	disp(xr)						8CXX	8DXX	8EXX	8FXX	Increment contents of effective address by 1; skip next instruction if result = 0; result is in EA; use address mode shown; XX = ±127
SUBB	0, disp(xr)						90XX	91XX	92XX	93XX	Subtract contents of effective address from AC0; result to AC0; use address mode shown; XX = ±127
JSR	@ disp(xr)						94XX	95XX	96XX	97XX	Jump to subroutine indirect; push PC onto stack; final address = to contents of location (XX + register shown); XX = ±127
JMP	@ disp(xr)						98XX	99XX	9AXX	9BXX	Jump indirect; final address = to contents of location (XX + register shown); XX = ±127
SKG	0, disp(xr)						9CXX	9DXX	9EXX	9FXX	Compare AC0 with contents of location (XX + register shown); XX = ±127; skip next instruction if AC0 > (EA)
LD	0, @ disp(xr)						A0XX	A1XX	A2XX	A3XX	Load indirect; load AC0 with contents of final address; address = contents of location (XX + register shown); XX = ±127
OR	0, disp(xr)						A4XX	A5XX	A6XX	A7XX	OR AC0 with contents of location (XX + register shown); XX = ±127; result to AC0
AND	0, disp(xr)						A8XX	A9XX	AAXX	ABXX	AND AC0 with contents of location (XX + register shown); XX = ±127; result to AC0
DSZ	disp(xr)						ACXX	ADXX	AEXX	AFXX	Decrement contents of effective address by 1; skip next instruction if result = 0; result is in EA; address = (XX + register shown); XX = ±127
ST	0, @ disp(xr)						B0XX	B1XX	B2XX	B3XX	Store indirect; store AC0 into final address; address = contents of location (XX + register shown); XX = ±127
SKAZ	0, disp(xr)						B8XX	B9XX	BAXX	BBXX	AND AC0 with contents of location (XX + register shown); skip next instruction if result = 0; XX = ±127
LSEX	0, disp(xr)						BCXX	BDXX	BEXX	BFXX	Load AC0 with sign extended; Bit 7 of location (XX + register shown) is extended to AC0 8-15; Bits 0-7 are loaded to AC0 Bits 0-7; XX = ±127
LD	r, disp(xr)		╳				C0XX	C1XX	C2XX	C3XX	Load AC0 with contents of location (XX + register shown); XX = ±127
				╳			C4XX	C5XX	C6XX	C7XX	Load AC1 with contents of location (XX + register shown); XX = ±127
					╳		C8XX	C9XX	CAXX	CBXX	Load AC2 with contents of location (XX + register shown); XX = ±127
						╳	CCXX	CDXX	CEXX	CFXX	Load AC3 with contents of location (XX + register shown); XX = ±127
ST	r, disp(xr)		╳				D0XX	D1XX	D2XX	D3XX	Store AC0 to location (XX + register shown); XX = ±127
				╳			D4XX	D5XX	D6XX	D7XX	Store AC1 to location (XX + register shown); XX = ±127
					╳		D8XX	D9XX	DAXX	DBXX	Store AC2 to location (XX + register shown); XX = ±127
						╳	DCXX	DDXX	DEXX	DFXX	Store AC3 to location (XX + register shown); XX = ±127
ADD	r, disp(xr)		╳				E0XX	E1XX	E2XX	E3XX	Add AC0 to location (XX + register shown); XX = ±127; result to AC0
				╳			E4XX	E5XX	E6XX	E7XX	Add AC1 to location (XX + register shown); XX = ±127; result to AC1
					╳		E8XX	E9XX	EAXX	EBXX	Add AC2 to location (XX + register shown); XX = ±127; result to AC2
						╳	ECXX	EDXX	EEXX	EFXX	Add AC3 to location (XX + register shown); XX = ±127; result to AC3
SKNE	r, disp(xr)		╳				F0XX	F1XX	F2XX	F3XX	Compare AC0 to location (XX + register shown); XX = ±127; if not equal skip next instruction
				╳			F4XX	F5XX	F6XX	F7XX	Compare AC1 to location (XX + register shown); XX = ±127; if not equal skip next instruction
					╳		F8XX	F9XX	FAXX	FBXX	Compare AC2 to location (XX + register shown); XX = ±127; if not equal skip next instruction
						╳	FCXX	FDXX	FEXX	FFXX	Compare AC3 to location (XX + register shown); XX = ±127; if not equal skip next instruction

National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051
(408) 737-5000
TWX: 910-339-9240

National Semiconductor GmbH
808 Fuerstenfeldbruck
industriestrasse 10
West Germany
Telephone: (08141) 1371
Telex: 05-27649

NS Electronics (HK) Ltd.
4 Hing Yip Street, 11th Floor
Kwun Tong
Kowloon, Hong Kong
Telephone: 3-411241-8
Telex: 73866 NSE HK HX

NS International Inc.
Miyake Bldg. 6F, 1-9 Yotsuya
Shinjuku-Ku
Tokyo 160, Japan
Telephone: 03-355-3711
Telex: J28592

NS Electronics Pty. Ltd.
CNR-Stud Road & Mountain Highway
Bayswater, Victoria 3153, Australia
Telephone: 03-729-6333
Telex: 32096

PUBLICATION NO. 420305292-001A, ORDER NO. IPC-16A/928
©1977 NATIONAL SEMICONDUCTOR CORPORATION
PRINTED IN U.S.A. DA-DD10M37